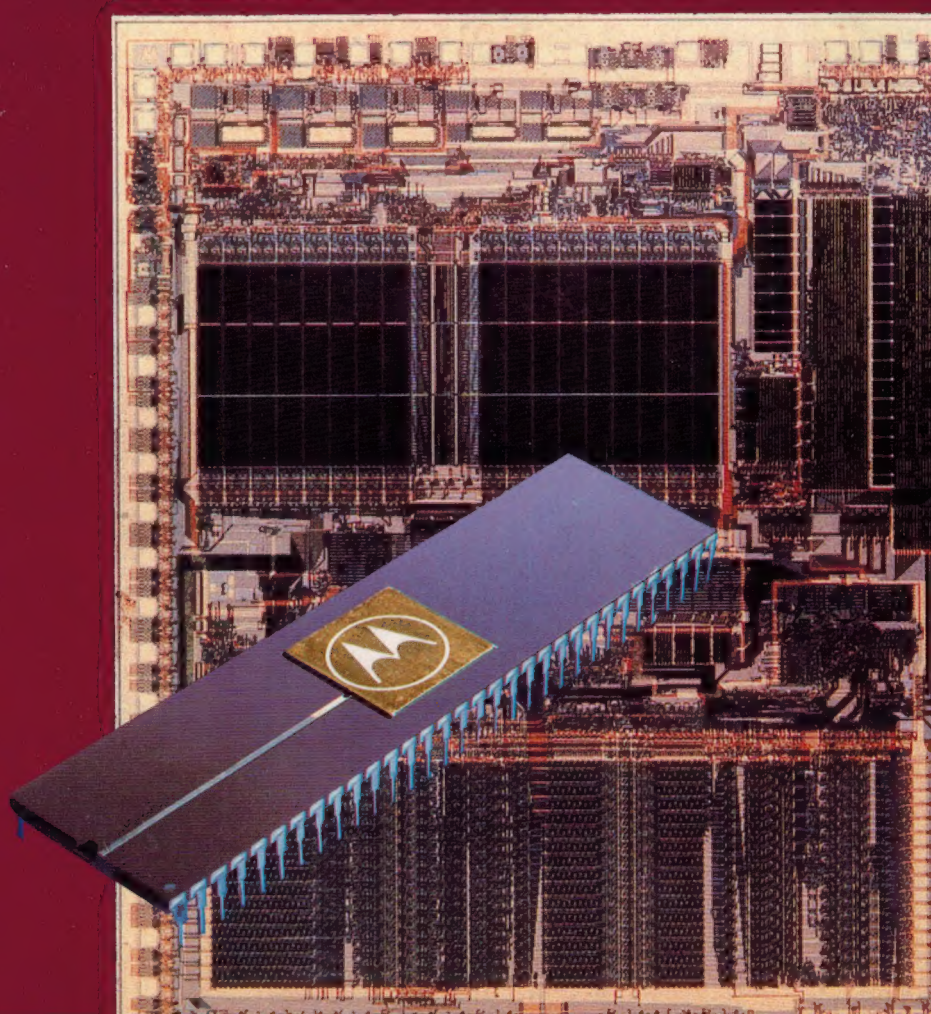


図解

16ビット マイクロコンピュータ

MC 68000 の使い方

小島 進 著 ● オーム社



図解 16ビットマイクロ コンピュータ8086の使い方

(A 5判 190頁)

井出裕巳 著

16ビットマイクロコンピュータの中で最も多く使われているインテル社の8086について、概要から各種命令、動作、プログラミングなどのポイントを、具体的かつ平易に解説しました。

図解 マイクロコンピュータ Z-80 の使い方

(A 5判 190頁)

横田英一 著

本書は、Z-80の使い方についてCPUの概要から各種命令、動作、プログラムのテクニックに至るまでを66項目に分類・整理し、初心者にもわかるようやさしく解説しました。

図解 マイコンの基礎知識

(A 5判 250頁)

矢田光治 著

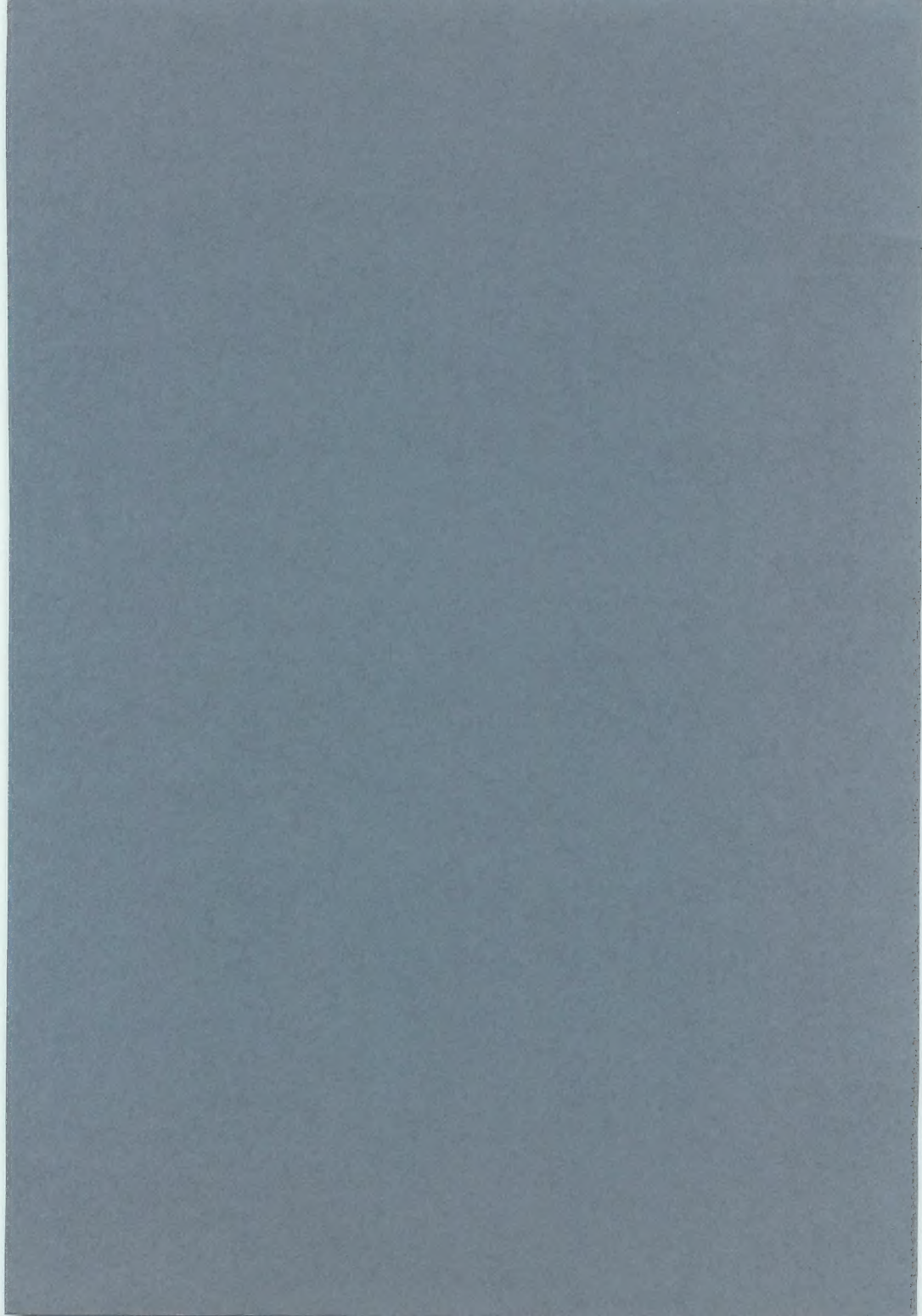
マイコンの基礎知識を94項目ページ単位に要約、2色刷で視覚的に解説しました。初心者にはマイコン事典として、専門家にはポイントの整理に役立ちます。

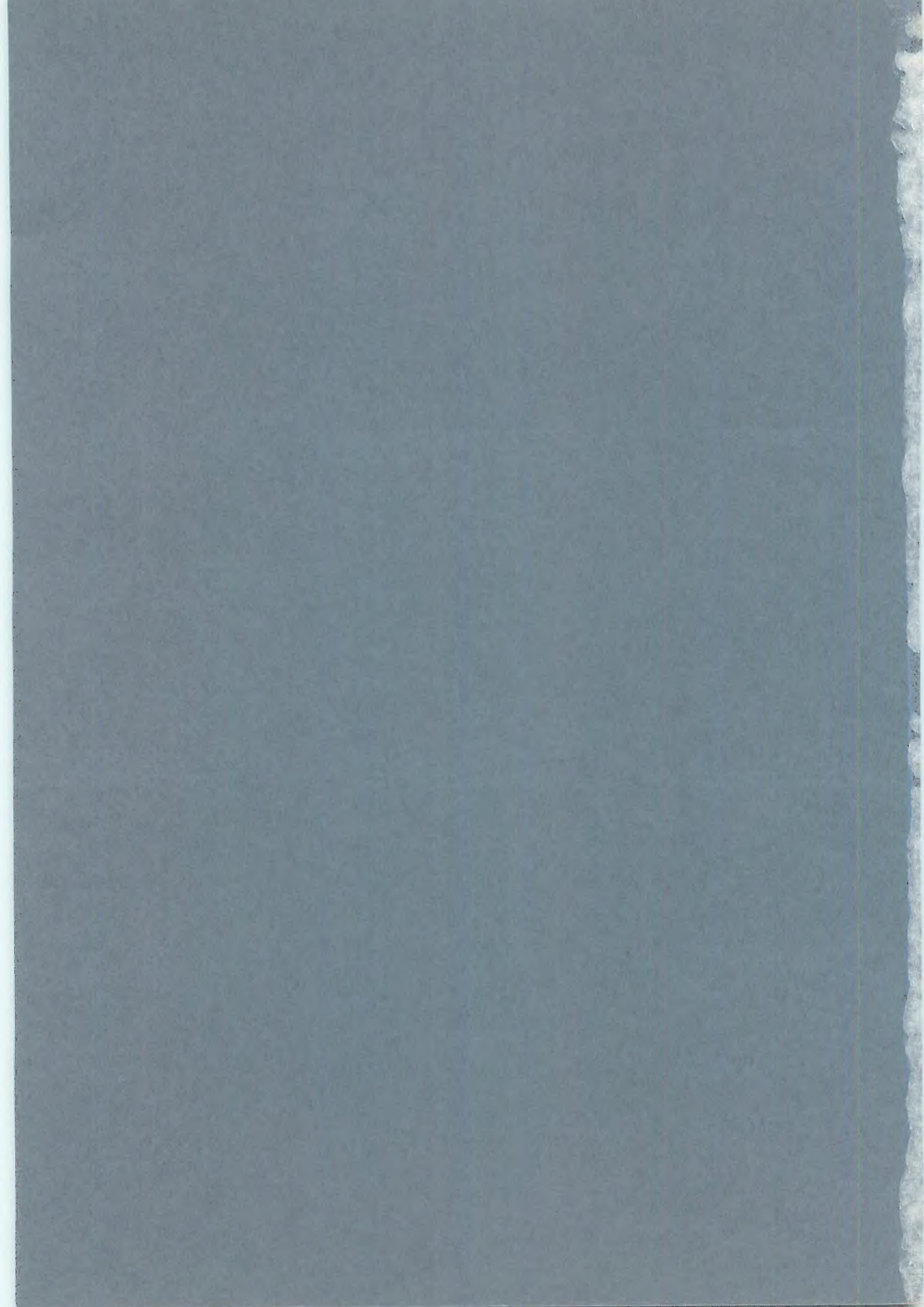
制御用マイコンの 作り方・使い方

(B 5判 240頁)

北川一雄 著

本書は、8085、Z-80系のマイコンを用いた各種制御回路・装置の作り方と、制御用マイコンの使用例を基礎と応用に分け、具体的プログラム例を示しました。





図解

16ビット マイクロコンピュータ

MC 68000

の使い方

小島 進 著

オーム社

本書は、「著作権法」によって、著作権等の権利が保護されている著作物です。

本書の全部または一部につき、無断で次に示す〔 〕内のような使い方をされると、著作権等の権利侵害となる場合がありますので御注意ください。

〔転載、複写機等による複写複製、電子的装置への入力等〕

学校・企業・団体等において、上記のような使い方をされる場合には特に御注意ください。

お問合せは下記をお願いします。

〒101 東京都千代田区神田錦町 3-1 Tel. 03-3233-0641

株式会社 オーム社出版局（著作権担当）

は し が き

マイクロプロセッサが出現し、1ビット、4ビット、8ビットとそれぞれの過程と経験を経て、より優れた能力が追求されて出現したのが16ビット・マイクロプロセッサであり、私たち人間の限りなき挑戦の成果のたまものであると思います。メモリ容量の増大、命令数の増大、命令実行スピードの速さなど、その技術進歩は急激であり、取り扱う私たちがその技術の吸収に追従しえないほどのすさまじさであります。

16ビット・マイクロプロセッサは、ハードウェア面においては、ミニコンの領域を越えて汎用小型・中型計算機に匹敵する能力を備えているといっても過言でないほどの強力な素子といえましょう。

また、だれの目にも明らかに映るのは、外形の大きさに比べその性能が驚異的なことであり、従来の小規模集積回路（SSI）や中規模集積回路（MSI）に比べ、集積度やプロセス技術が数倍に飛躍していることです。私たち人間の限りなき技術への挑戦と、それをかちえてゆく技術者に賞賛と敬意を払います。

現在、第2世代の16ビット・マイクロプロセッサの時代に突入したわけですが、8ビット・マイクロプロセッサにとって代わって、その隆盛を見るためには、よりいっそうのソフトウェアの充実が課題でありましょう。また、8ビットから16ビットへと移行するにつれて、デバイス価格もどんどん低減してゆき、それが引金となって種々の分野へと需要が広がってゆくと思われます。また、第3世代の32ビット・マイクロプロセッサの時代も目前に控えており、より充実したソフトウェアの開発は、今後よりいっそう要求されるものと思われます。

以上のような背景で、本書は16ビット・マイクロプロセッサ、MC 68000とはどうゆうものかを理解していただくことに重点を置いて執筆いたしました。また、紙数の都合上、68000とのインタフェース関係、プログラミング関係、アプリケーション関係の執筆が不足しておりますが、別の機会に紹介できればと考えております。

は し が き

各引用資料として、本書の末尾に参考資料を掲載しておきましたので、読者諸氏が MC 68000 の検討や導入に際してご参照くだされば、よりいっそうのご理解を得ていただけるものと信じます。

なお、本書執筆にあたり、ご協力いただいた日本モトローラ(株)伊南恒志氏、および末尾の参照資料、写真などの引用のご承諾をいただいた日本モトローラ(株)広報課、マーケティング課のご協力に深く感謝いたします。また、本書の編集、出版に際して、オーム社出版部の方々に大変なお世話をいただき、この出版ができたことに厚くお礼申し上げます。

昭和 58 年 8 月

著 者 し る す

目 次

1. 16ビット・マイクロプロセッサ入門

- 1・1 MC 68000 の誕生まで..... 2
- 1・2 8ビットMPU から16ビットMPUへ 4
- 1・3 MC 68000 の特徴..... 8

2. MC 68000 の構成

- 2・1 ピ ン 構 成16
- 2・2 レジスタの構成18

3. メモリの構成

- 3・1 メモリの構成24
- 3・2 メモリ内のデータ構成と使用26
- 3・3 スタックの構成28
- 3・4 キューの構成30

4. 入/出力の構成

- 4・1 バス・シグナル34
- 4・2 バス・コントロール・シグナル36
- 4・3 MPU コントロール・シグナル.....44
- 4・4 MPU ステータス・シグナルとクロック.....48
- 4・5 データ転送50

5. アドレッシング・モード

- 5・1 実効アドレス54

目 次

5・2 レジスタ直接モード	56
5・3 レジスタ間接モード	58
5・4 絶対（アブソリュート）モード	64
5・5 プログラム・カウンタ相対モード	66
5・6 イミディエート・モード	68
5・7 インプライド・モード	70

6. 例 外 処 理

6・1 例 外 処 理	74
6・2 例外処理シーケンス	84
6・3 パス・エラー	86
6・4 リセットとトラップ	90
6・5 特 権 状 態	92
6・6 特権違反とトレース	94

7. 命 令 セ ッ ト

7・1 命令形式と命令タイプ	100
7・2 データ移動命令	102
7・3 整数算術演算命令	107
7・4 論理操作命令	113
7・5 シフトとローティット命令	115
7・6 BCD 演算命令	119
7・7 ビット操作命令	121
7・8 プログラム制御命令	124
7・9 リンク命令とアンリンク命令	129
7・10 システム制御命令	132

8. システムの構成

8・1 最小システム	136
------------	-----

8・2 中位システム構成	138
8・3 上位システム構成	139
8・4 システムと VERSAbus の関係	140
9. 周辺ファミリ・チップ	
9・1 MC 68000 をサポートする周辺チップ	146
9・2 メモリ管理ユニット MC 68451	148
9・3 DMA コントローラ MC 68450	152
9・4 MC 68000 周辺通信用 LSI	157
9・5 MC 68000 と MC 6846 のインタフェース例	159
10. MC 68000 開発装置	
10・1 エクサマクス	168
10・2 エクサマクスの使用例	175
10・3 VMC 68/2 マイクロコンピュータ	181
11. 開発ソフトウェア/リアルタイム・モニタ	
11・1 VERSAdos 開発ソフトウェア	188
11・2 RMS 68 K リアルタイム・モニタ	197
12. MC 68000 の将来の発展方向	
12・1 MOS 技術の方向	210
12・2 ソフトとハード・コストの方向	211
12・3 MC 68000 の発展方向	212
付 録	
付録 1. レジスタ転送言語定義	215
付録 2. MC 68000 アセンブラ命令一覧表	217
付録 3. オペレーション・コード表	223

目 次

付録 4. 電氣的仕様とタイミング図	228
参 考 文 献	237
索 引	238

1. 16ビット・マイクロ プロセッサ入門

16ビット・マイクロプロセッサが作られるまでの背景と、モトローラ社のMPUの推移を説明し、8ビットMPUと16ビットMPUの比較および16ビットの特長を記しています。また、それぞれの開発支援装置の推移も説明しています。

1・1 MC 68000 の誕生まで

マイクロコンピュータは、約10年前に発表されて以来、各方面のニーズにこたえるべく、技術革新とともに急速な進歩と発展を遂げてきています。モトローラ社が8ビット・マイクロプロセッサ MC 6800 を発売した年は1974年であり、16ビット・マイクロプロセッサ MC 68000 が誕生するまでには、その後5年の年月が費やされ、1979年に発売されて今日に至っております。また他社においては、爆発的人気の出たインテルの8080が1973年に発売され、そしてザイログのZ80が1976年に次々と発売され、今日の8ビット・マイクロプロセッサの隆盛を見るに至っております。

年月の経過とともに用途、方法も年々向上し、応用範囲も広がり、より高速な数値演算やマルチプログラミングによるメモリ管理方法、膨大なメモリを必要とするデータ処理方法など、より高度な使用方法や要求が今日の8ビット・マイクロプロセッサの能力を超えるものとなり、16ビット・マイクロプロセッサが使用されるようになった背景です。また製造面についても、回路技術の進歩、エッチング方法の進歩、プリンティング方法の進歩など大きな技術的躍進がなされたことです（1・2節「8ビットMPUから16ビットMPUへ」参照）。

モトローラのMPUの超LSI回路技術は、図1・1に示すように、発展方向は二つの方向をたどりました。機能面が強化された方向として、MC 6808, MC 6802, MC 6801などのマイクロプロセッサがあり、効率面が強化された方向としては、MC 68A00, MC 68B00, MC 6809などのマイクロプロセッサがあります。

68000は、このようなMC 6800系でのアーキテクチャの基盤と集積度技術の向上、そしてプロセス技術の向上などの蓄積と統合によって生まれてきたのです。その応用分野は、従来の特殊分野から脱皮し、OA関係、ロボット関係の工場や各機器の自動化、民主機器関係の導入へと広がり、16ビット・マイクロプロセッサ主流の時代に突入しようとしています。また一部、第3世代の32ビットMPUの要望もすでに出ており、モトローラ社の上位に位置する仮想MPU, MC 68020,

32ビットMPUも1983年に発売が予定されています。

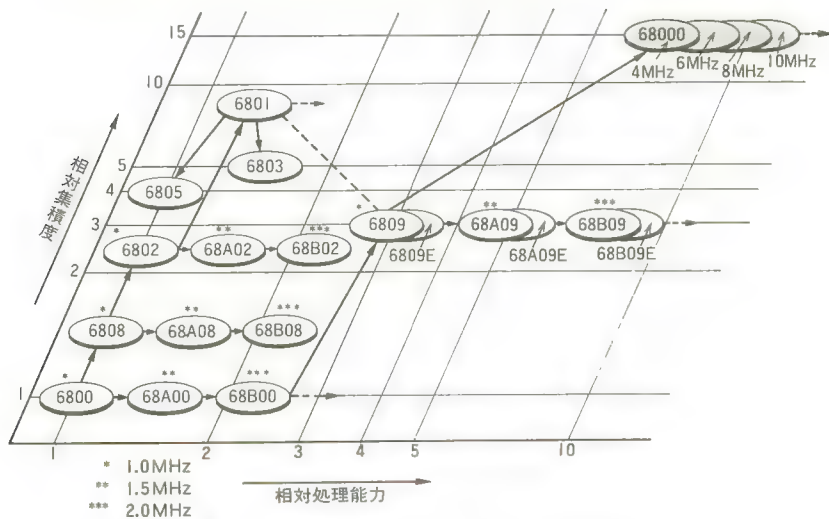


図 1・1 モトローラ・マイクロプロセッサの歩み

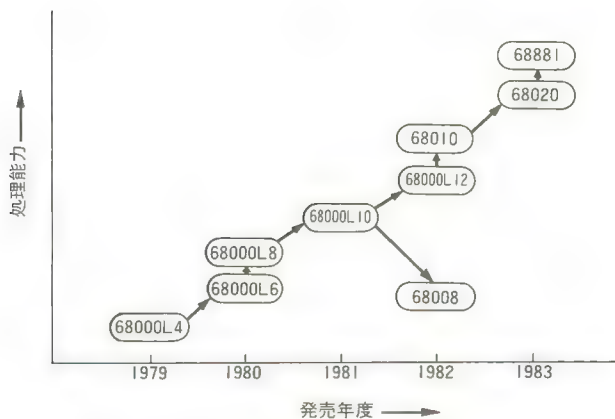


図 1・2 16/32ビット・マイクロプロセッサ関連図

1.2 8ビット MPU から 16ビット MPU へ

1.1 章で述べたように、16ビット・マイクロプロセッサは、時代のニーズにこたえるべく必然的に生まれたことは明かです。ロボット制御に用いるより精度の高い座標軸計算や、大容量のメモリを必要とする画像処理とか、日本語処理に便利な16ビットのMPUが、本格的に検討や応用がなされ、普及しています。

16ビットMPU 68000は、従来の8ビットMPU 6800とのコンパチビリティおよび一貫性を保ちながらのアーキテクチャや超LSI回路技術、プロセス技術の改良と開発がなされたことです。また、サプライヤが市場の要求に応じうる十分な供給と適応できる価格が可能であったことも、普及しえた大きな要因の一つといえましょう。

〔1〕 16ビットになって改善された点

(1) 多数のレジスタとスタック、広いアドレス・レンジ、高級言語指向命令(LINK, UNLK, CHKなど)による高級言語の支援。

(2) オペレーティング・システム(OS): 特権命令、メモリ管理、ベクタ多重レベル割込み、トラップ構造および特定の命令(MOUER, MOUEM, TRAPなど)による支援。

(3) マルチプロセッサ・システム: ハードウェア・インタロックとソフトウェア・インタロックを備える。

フラグは、8ビットMPU 6800のときはコンディション・コード・レジスタといわれ、MPUの演算の状態(一部マスクビットを含む)を示すだけであったが、16ビットMPU 68000ではステータス・レジスタに変更されて16ビットに拡張され、MPU全体のステータスを示すフラグとなり、システム・モード、ユーザ・モード、割込み受け付け、割込みレベルなどを備え、将来のMC 68000のファミリの拡張に備えて十分広いスペースをとっています(図2.4参照)。

〔2〕 命令シーケンスと処理速度の代表的改善点 クイック・イミディエート・アドレッシング・モードを用いたシングル・ワード・オペレーションの加算や減算命令は、データ・レジスタとメモリの処理速度を速めました。MOVEQ

I 16ビット・マイクロプロセッサ入門

(move quick) 命令では、シングル・ワード・オペレーションで符号付きバイト・データを任意のレジスタにロードできます。また、DBcc (decrement and branch) 命令ループは、オペレーションの処理速度を速めるために、一つの命令でコンディションのテストを行い、デクリメント分岐ができます。また、コード減らしや演算オペレーションの性能を高める命令として、MULS (signed multiply), MULU (unsigned multiply), DIUS (signed divide), DIVU (unsigned divide) があり、演算オペレーション命令として ABCD, SB CD などがあります。

ペリフェラル・コントロールとのインタフェースについても、MC 68000 と 68 系 8 ビット MPU に開発された周辺デバイスが直接インタフェースできるようになっています。シグナル \overline{E} (enable), \overline{VMA} (valid memory adress), \overline{VPA} (valid peripheral adress) などが設けられています (図 2・5, 図 4・12 参照)。

表 1・1 16ビット・マイクロプロセッサ製造会社

タイプ	海 外	日 本
MC 68000	モトローラ モステック、フィリップス ロックウェル、シグネテックス	日 立
I 8086	インテル AMD ハリス シーメンス	日本電気 富士通 三菱電機 東 芝
Z 8000	ザイログ AMD	シャープ 東 芝
TMS 9995	テキサス・インスツルメント	—
NS 16000	ナショナル・セミコンダクタ	—
F 9445	フェアチャイルド	—

表 1・2 8ビット MPU と 16ビット MPU 比較

区 分	6800	68000
集積度	NMOS	HMOS
エッチング	ケミカル・エッチング	ドライ・プラズマ・エッチング
プリンティング	ダイレクト・コンタクト・プリンティング	プロジェクション・プリンティング
ブロック・ゲート面積	4 128 μm^2	1 852 μm^2
速度電力積	4 ピコジュール	1 ピコジュール

I 16ビット・マイクロプロセッサ入門

〔3〕 開発支援装置 開発支援ツールについても、図1・3に示すように8ビット MPU から16ビット MPU に移るとともに各装置が出ております。8ビット MPU 用エクササイザ (EXORciser) に始まり、エクサセット (EXOR set), 16ビット MPU 用のエクサマクス (EXOR macs), そしてボックス・コンピュータ (VMC 6812) と MC 68010 VM プロセッサ搭載のマイクロコンピュータシステム (VME/10) のような推移と種類の拡大が行われてきています。

なお、今後とも引き続きマイクロプロセッサの発展とともに、各種各様の多くの開発支援装置が生まれてくると思います。

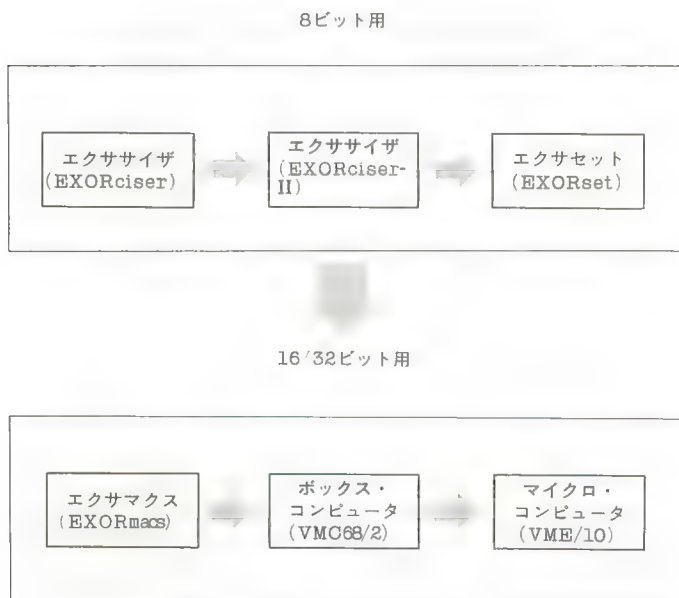


図 1・3 開発支援装置の移行

1 16ビット・マイクロプロセッサ入門



図 1・4 モトローラ MPU 開発支援装置

1.3 MC68000の特徴

MC68000 は、16/32 ビット・マイクロプロセッサとして設計されており、クロック周波数は 4 MHz, 6 MHz, 8 MHz, 10 MHz, 12 MHz の各バージョンが用意されております。また、32 ビットのレジスタを 16 個持ち、16 ビットのデータ・バス構成をとり、24 ビットのアドレス・バスを持ちます。ページングやセグメンテーションの必要がなく、16 M バイトの直接アドレスが可能です。アーキテクチャにおいても、一貫思想のもとにメモリ・マップド I/O が忠実に受け継がれており、メモリと I/O を意識することなく同等に扱えることです。

オペレーティング・システム用の機能として、ソフトウェア・トラップ命令、ハードウェア・トラップ機能、スーパーバイザ/ユーザの各モードを持ち、7 レベルの外部割込みに対応できる 192 のユーザ・インタラプト・ベクタがあります。バスは、同期/非同期の両方式が使用でき、従来の 8 ビット MPU ファミリの LSI を直接接続できます。

68000 の主な特長は次のような点です。

(1) 系統立った一貫した構造のアーキテクチャ

- ・プログラムの記述は、高級言語で記述するのと同様な容易さでアセンブラ言語が使用できます。
- ・フレキシビリティと能力を大幅に高め、整数データのオペランド操作におけるデータ・レジスタとメモリ・ロケーションは、ソースにも行先にも自由に指定できます。
- ・アドレッシング・モードは、ニーモニック（命令操作）から独立し、すべてのアドレス・レジスタをレジスタ直接、レジスタ間接、インデックス・アドレッシング・モードで使用できます。

(2) 構造化モジュラ・プログラム

- ・ブロック構造高級言語（構造化アセンブラや PASCAL など）が使用できます。
- ・総計 255 のベクタ・ロケーションを持ち、これらの割込みをハードウェア・

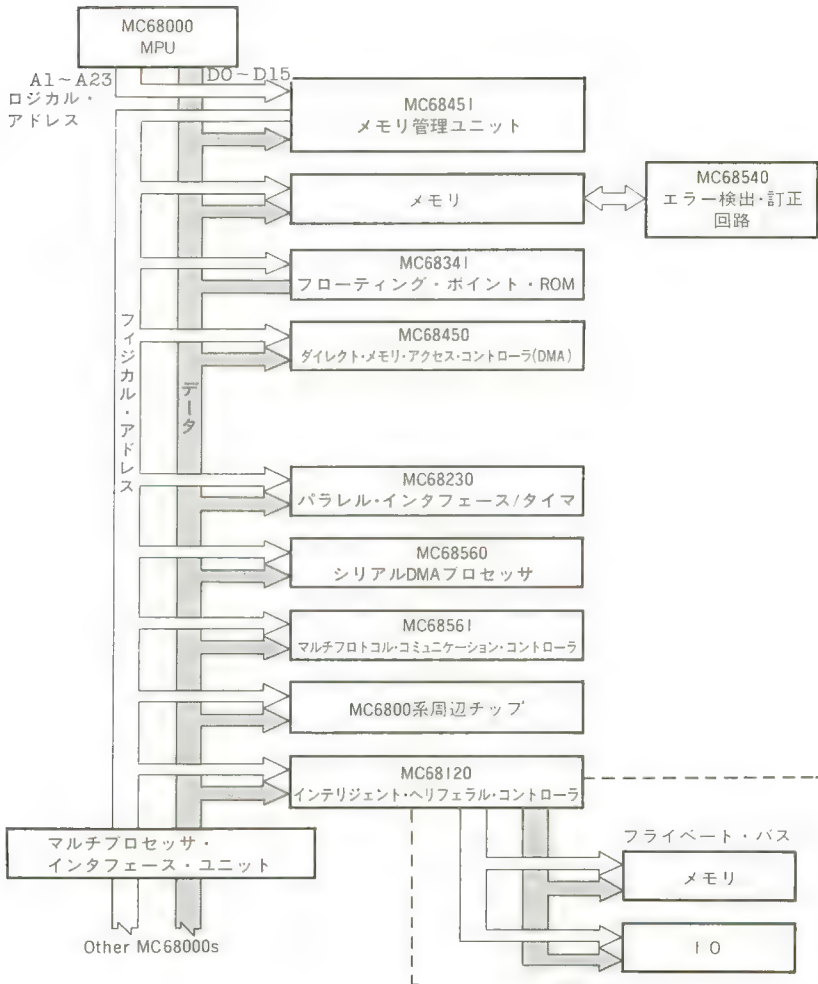


図 1・5 ローカル・バスの拡張

I 16ビット・マイクロプロセッサ入門

トラップ、ソフトウェア・トラップに割り当てています。

- ・サブルーチン・コールのオーバーヘッドを減少させるLINK, UNLK 命令、プログラムが指定するレジスタ群をアドレスの指定のみで一度に転送できるMOVE 命令などがあります。
- ・プログラミング技法をサポートする命令や、システム・コール・ルーチンやユーザのマイクロルーチン処理に便利なTRAP 命令などがあります。

(3) 大アドレス空間のメモリ管理

- ・16M バイトのアドレス空間は、ワードやバイト単位で直接アクセスできます。
- ・メモリ管理をソフトウェアで行うときのCHK 命令は、メモリ保護や管理ができます。
- ・ユーザ状態とスーパーバイザ状態の2種の区別があり、スーパーバイザ状態では、プロセッサ・システム内で保護されている数種の実操作が可能であり、ユーザ状態では、外部メモリ管理ユニットを利用して、広いアドレス・レンジが使用できます。

(4) ソフトウェアのテスト機能の強化

- ・システム・プログラムにおけるプログラム・エラーと虫(BUG)の検出機能が強化されました。
- ・プログラム作成において、16 個のソフトウェアTRAP 命令を用いてエラー検出と訂正ルーチンが自由に作成できます。
- ・一つ一つの命令ごとにプログラムのデバッグが行えるトレース機能を備えています。

(5) 将来へのフレキシビリティ

- ・マルチレベル・マイクロプログラム構造により、命令の実行にはフレキシビリティがあり、命令のオペコード・マップの1/8 以上が将来の命令追加のために特別に確保されています。
- ・ユーザは、現在の命令セットにない命令をトラップ命令とエミュレータ・トラップを用いて作成、実行が可能です。

さらに現在では、MC68000 の上位機種も続々と発表される予定です。バーチャルメモリ、バーチャルI/O を強力にサポートするMC68010, 32 ビットMPU の

システム・インタラプト

MC68000の割込みレベルは7レベルありますが，“スピードマスタ68K”では，そのうちの6個を使用しています．

下表に，割込みレベルの各デバイスへの割合を示します．

表 1・3 割込みレベル

レベル	デバイス
1	未使用
2	PI/Tタイマ
3	PI/Tパラレル・ポート
4	M6800 インタフェース*
5	ACIA 1*
6	ACIA 2*
7	アボート・スイッチ*

* オートベクタ

メモリ・マップ

表 1・4 メモリ・マップ

	アドレス	資 源
ROM領域	S000000-S000007*	ROM/EPROM [イニシャル・スタック・ポインタ]
RAM領域	S000008-S0007FFF	システム RAM [イニシャル・プログラム・カウンタ]
	S000800-S007FFFF	ユーザ RAM
ROMファームウェア	S008000-S00BFFFF*	ROM/EPROM
	S00C000-S00FFFFF	使用していない
	S010000-S01003F	PI/T (下位バイト)
	S010040-S010043	ACIA 2 (下位バイト)
		ACIA 1 (上位バイト)
I/O デバイス	S010044-S01007F	拡張用 ACIA
64K バイト・ページ	S010080-S0100BF	拡張用 PI/T
	S0100C0-S0100FF	拡張用 ACIA
	S01100-S01FFFF	拡張用 ACIA と PI/T
		0000 0001 xxxx xxxx x0xx xxx0 PI/T
		0000 0001 xxxx xxxx x0xx xxx1 使用せず
		0000 0001 xxxx xxxx x1xx xxx0
		ACIA 2
		0000 0001 xxxx xxxx x1xx xxx1
		ACIA 1
		使用していない
		6800 64K byte ページ [周辺デバイスのインタフェース用]
		使用していない
	S020000-S02FFFF	
特別なデコード信号	S030000-S03FFFF	
	S040000-S04FFFF	

* リードのみ

図 1・6 スピード・マスタ68Kの割込みレベルとアドレス・デコード

I 16ビット・マイクロプロセッサ入門

MC68020、浮動小数点演算用コ・プロセッサなどがあります。また、内部がMC68000と完全にコンパチブルで、外部バスのみ8ビット構成で、中規模システムのハードウェア・コスト低減とソフトウェア向上の用途としてMC68008などがあります(12・3節参照)。

16ビット・マイクロプロセッサMC68000の入門用や学習用として、安価な単一ボード・コンピュータ“**SPEED MASTER 68K**”を紹介します。

図1・7にスピード・マスタの実線図を示します。また、図1・8にブロック図を示します。

ファームウェア：ファームウェアはプログラム作成とオペレーションを制御し、2個の8KバイトROMに格納されています。そして、ユーザにデバッグ/モニタ機能、プログラム入力、アセンブラ、逆アセンブラ、I/O機能を提供します。

デバッグ/モニタ：他のモトローラ製品と互換性を持たせるため、用意されているコマンドはすべてMACSbug、VERSAbugのコマンド形式と同一です。デバッグ機能には、メモリ表示/変更、レジスタ表示/変更、プログラム・トレース、ブレーク・ポイント、プログラム実行、データ変換、トランスパーレント・モードなどがあります。

アセンブラ/逆アセンブラ：ソース・プログラムが直接ファイル化されないダイナミックなアセンブラ、エディタ・プログラムです。各命令は1行入力するごとに適切な機械語に翻訳され、メモリへ格納され、翻訳された機械語を逆アセンブルし、ニーモニックとオペランドを表示します。

I/O機能：端末を介してのユーザ・インタフェースのサポートに加え、ファームウェアは他のI/O機能もサポートします。ホスト・コンピュータからのオブジェクト・プログラムの転送やオーディオ・カセットへのオブジェクト・プログラムの格納と取出しやプリンタの制御などを支援します。

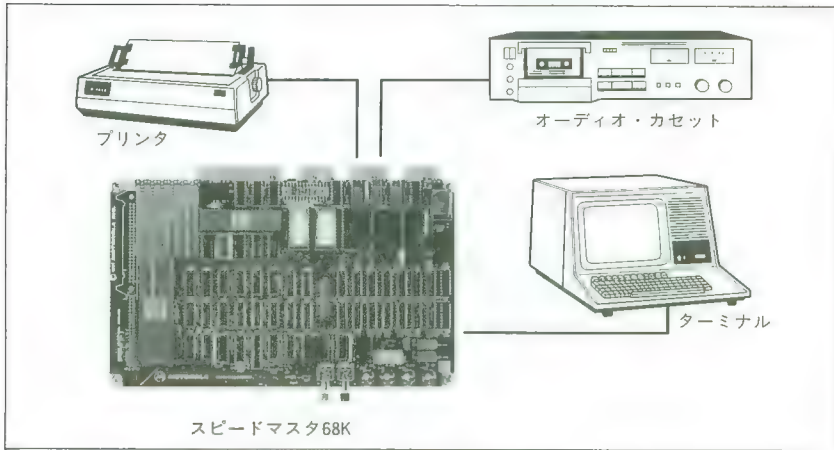


図 1.7 スピード・マスタ 68K 図

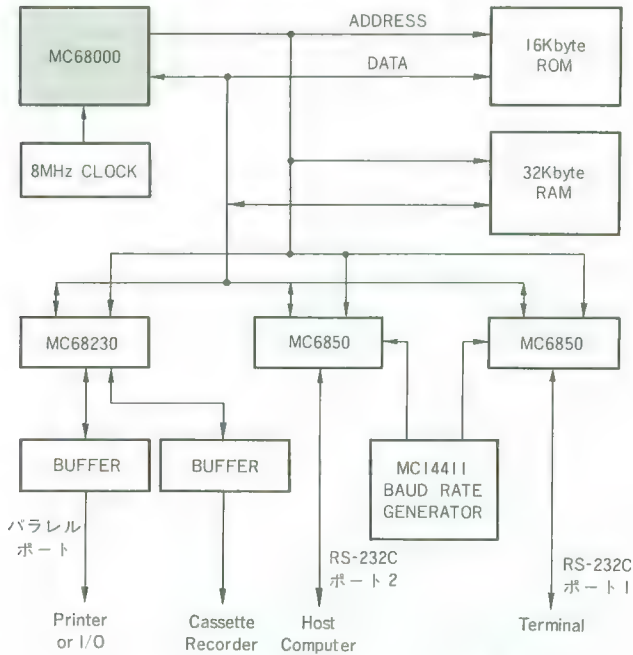


図 1.8 スピード・マスタ 68K ブロック図

〔マ ト メ〕

1. ハードウェアの特長

- 1) 32ビット長のデータ・レジスタ
- 2) 32ビット長のアドレス・レジスタ
- 3) 16M バイト直接アドレス空間
- 4) 56種の強力な基本インストラクション
- 5) ビット, BCD, バイト, ワードおよびロング・ワードのデータ・タイプをサポート
- 6) 14種の強力なアドレッシング・モード
- 7) メモリ・マップド I/O
- 8) 同期/非同期のバス・サイクル

2. ソフトウェアの特長

- 1) オペレーティング・システム・サポート機能
- 2) 高効率で高級言語によるプログラム処理
- 3) クリーンで拡張性に富むストラクチャ

2. MC 68000 の構成

MC68000 のピン数やピン端子名の役割と機能について一覧表で説明し，内部におけるレジスタの数やレジスタ構成を図示し，おのこのレジスタの役割についての概要を説明します．

2 MC68000 の 構 成

2.1 ピ ン 構 成

MC68000 は、64 ピンの DIP (Dual Inline Package) に収められた HMOS (High Density Short Channel MOS) の VLS の IC です。

MC68000 のピン端子番号と信号の関係は図 2.1 に示します。また、シグナル名と機能は表 2.1 に示します。

各シグナルの詳細な機能については、第 4 章の入/出力の構成を参照してください。V_{cc} 電源は、+5V 単一電源（最小 2.4V）で動作します。また、MPU 内部のレジスタはダイナミック RAM と同様な動作になっていますので、データの取扱いとクロック周波数には配慮が必要です。

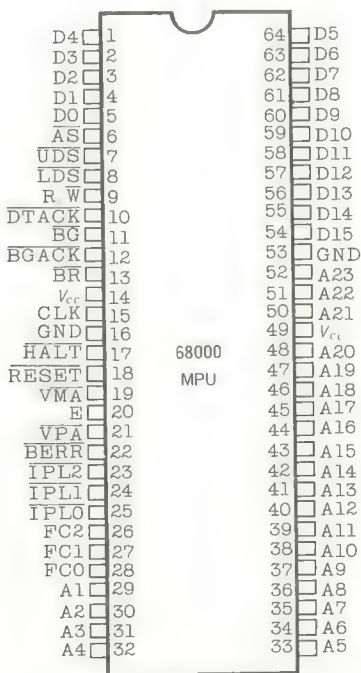


図 2.1 MPU とピン番号

2 MC68000 の 構 成

表 2・1 ピン番号と信号の関係

ピン番号	信 号 名	機 能	形 式
14	Vcc	電 源	入力
49	Vcc		
16	GRD	グラウンド	入力
53	GRD		
15	CLK	クロック	入力
5	D0	データ・ライン	入力/出力 (双方向 3) ステート
4	D1		
3	D2		
2	D3		
1	D4		
64	D5		
63	D6		
62	D7		
61	D8		
60	D9		
59	D10		
58	D11		
57	D12		
56	D13		
55	D14		
54	D15		
29	A1	アドレス・ライン	出力 (単方向 3) ステート
30	A2		
31	A3		
32	A4		
33	A5		
34	A6		
35	A7		
36	A8		
37	A9		
38	A10		
39	A11		
40	A12		
41	A13		
42	A14		
43	A15		
44	A16		
45	A17		
46	A18		
47	A19		
48	A20		
50	A21		
51	A22		
52	A23		
6	AS	アドレス・ストローブ	出力
7	UDS	アップ・データ・ストローブ	出力
8	LDS	ローア・データ・ストローブ	出力
9	R/W	リード/ライト	出力
10	DTACK	データ・トランスファ・アクノリッジ	入力
11	BG	バス・グラント	出力
12	BGACK	バス・グラント・アクノリッジ	入力
13	BR	バス・リクエスト	入力
17	HALT	ホルト	入/出力
18	RESET	リセット	入/出力
19	VMA	バリッド・メモリ・アクセス	出力
20	E	イネーブル	出力
21	VPA	バリッド・ペリフェラル・アドレス	入力
22	BERR	バス・エラー	入力
23	IPL2	インタラプト・コントロール	入力
24	IPL1		
25	IPL0		
26	FC2	プロセッサ・ステータス	出力
27	FC1		
28	FC0		

2・2 レジスタの構成

MC68000 の特徴である 32 ビット長の合計 15 個の汎用レジスタと 4 個のコントロール・レジスタは、仕様を十分に満たし、魅力あるチップといえましょう。また、演算レジスタとアドレス・レジスタが分離していることも特徴の一つといえましょう。

68000 のレジスタの構成を機能別に分類しますと、次の 5 種類に分けられます。

1) データ・レジスタ、2) アドレス・レジスタ、3) スタック・ポインタ、4) プログラム・カウンタ、5) ステータス・レジスタ。

[1] **データ・レジスタ** DO～D7 までの 8 個のデータ・レジスタを備え、1, 8, 16, 32 ビットのおおのこのデータ・オペランドをサポートします。また、演算や転送などの各命令オペランドサイズは、B=バイト (8 ビット)、W=ワード (16 ビット)、L=ロング・ワード (32 ビット) があります。そして、バイトやワードの下位ビットをソース・オペランドする場合や行先オペランドの場合は、該当部分が変わり、残り上位ビットは使われません。

[2] **アドレス・レジスタ** A0～A6 までの 7 個のアドレス・レジスタを備え、アクティブ・スタック・ポインタとともに、32 ビットのアドレス・オペランドをサポートします。A0～A6 は汎用インデックス・レジスタとして用い、アドレス・モードとともに全アドレス空間を柔軟に用いることができます。アドレス・レジスタのアクセスは、ワード (16 ビット) かロング・ワード (32 ビット) のどちらかをを用います。したがって、バイト・サイズのエペランドは使用できませんので注意してください。

[3] **スタック・ポインタ** A7 のスタック・ポインタは、16 ビットのユーザ・スタック・ポインタと 16 ビットのスーパーバイザ・スタック・ポインタの 2 種類を備え、どちらか一方の状態で動作します。また、スーパーバイザ状態でシステム・スタック・ポインタとして働き、トラップや割込み時のプログラム・カウンタとステータス・レジスタのスタックにも有効です。

2 MC68000 の 構 成

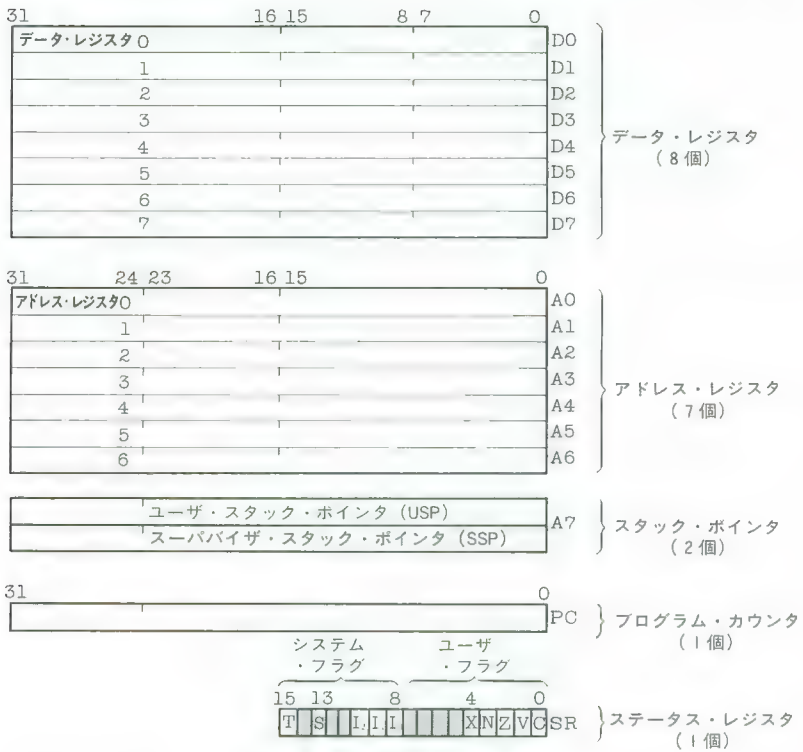


図 2・2 68000 のレジスタ構成

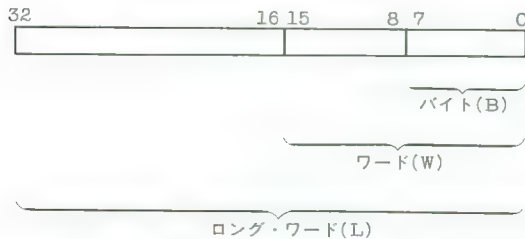


図 2・3 レジスタのデータ処理サイズ

2 MC68000 の 構 成

〔4〕 **プログラム・カウンタ (PC)** 32 ビットのプログラム・カウンタは、プログラム制御や実行アドレスのジャンプやデータ参照が行えます。32 ビット中 24 ビットが外部で使用できる範囲です。したがって、生成アドレスは \$00000000 ~ \$FFFFFFF までです。 こう見た方がいいと思う。

〔5〕 **ステータス・レジスタ (SR)** 16 ビットのステータス・レジスタはシステム・バイトとユーザ・バイトに分かれており、ユーザ・バイトの下位 5 ビット (ビット 0~4) をコンディション・コード (条件フラグ) ともいっております。

・C (キャリー) : Bit 0

(1) 加算でオペランドの最上位ビットからキャリー (桁上げ) が出ると “1” にセット。

(2) 減算でボロー (桁下げ) が出ると “1” にセット。

・V (オーバーフロー) : Bit 1

加減算によってオーバーフローがあれば “1” にセット。

・Z (ゼロ) : Bit 2

(1) オペレーションの結果が “ゼロ” の場合に “1” にセット。

(2) ゼロ以外のときは Z フラグは “0” にクリアされる。

・N (ネガティブ) : Bit 3

(1) 8 (バイト), 16 (ワード), 32 (ロング・ワード) の演算結果の最上位ビットが “1” (負) であった場合に “1” にセット。

(2) それ以外の場合、N フラグは “0” にクリアされる。

・X (エクステンド) : Bit 4

(1) 加減算、ネゲート、シフトなどでキャリー (C) ビットと同じように変化し、“1” にセットする。

(2) データ移動時には変化しない。

・I₀, I₁, I₂ (割込みマスク) : Bit 8~10

割込み優先順位をこの 3 ビットの構成で決定し、優先順位は 7 が最も高く、1 が最下位です (表 2.2 参照)。

・S (スーパバイザ) : Bit 13

(1) S=1 のときスーパバイザ・ステート (特権状態で、上位にある)。

2 MC68000 の 構 成

(2) $S=0$ のときユーザ・ステート (特権状態でも下位に位置します)。

・T (トレース・モード) : Bit 15

$T=1$ のときデバッグ機能によりシングル・ステップでテスト中のプログラムの実行をモニタできます。

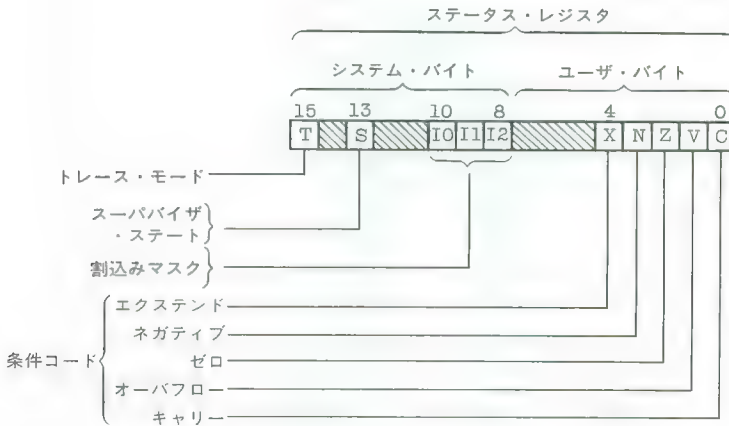
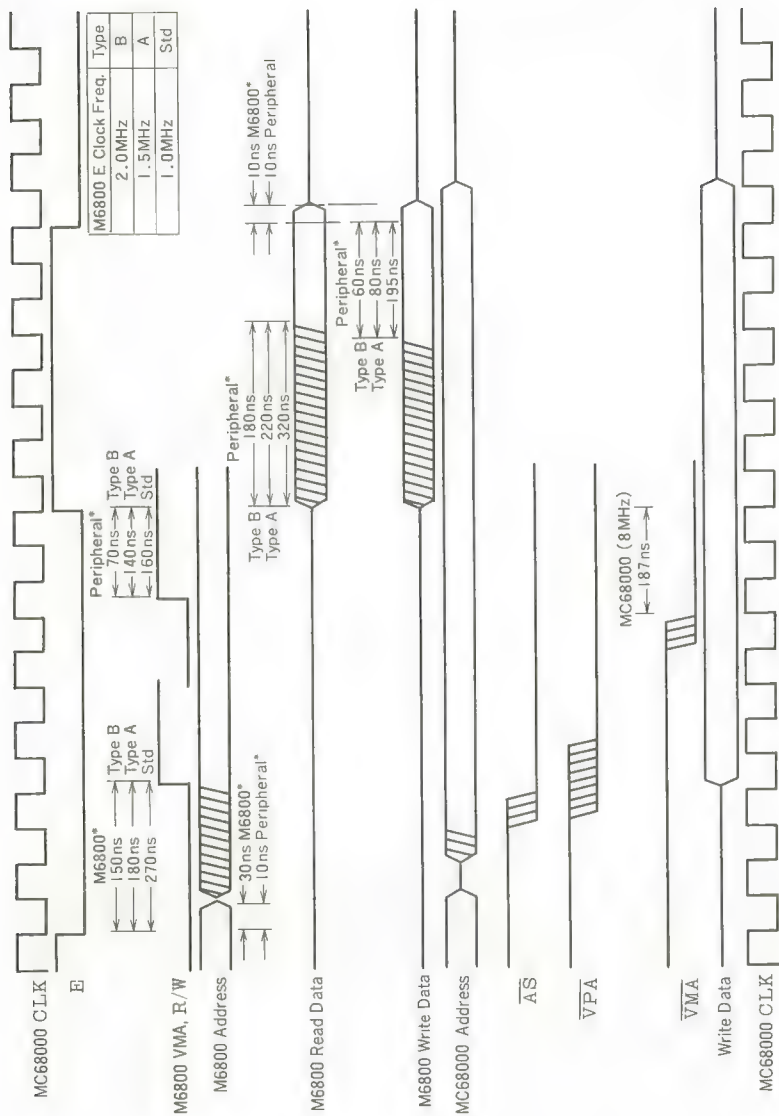


図 2・4 ステータス・レジスタ

表 2・2 割込みマスクとオートベクタの関係

割込みマスク	I_2 I_1 I_0	割込み優先度	割込み内容	ベクタ番号	ベクタアドレス (Hex)
7	1 1 1	1位	LEVEL7 INTERRUPT AUTO VECTOR	31	07C
6	1 1 0	2位	LEVEL6 "	30	078
5	1 0 1	3位	LEVEL5 "	29	074
4	1 0 0	4位	LEVEL4 "	28	070
3	0 1 1	5位	LEVEL3 "	27	06C
2	0 1 0	6位	LEVEL2 "	26	068
1	0 0 1	7位	LEVEL1 "	25	064



* Times are expressed for different device clock frequencies

図 2・5 68000 と 6800 周辺デバイスをインタフェースしたときのタイミング

3. メモリの構成

MC68000 が持つ 16M バイトのメモリ構成や、メモリの区分、割付けについて述べ、そのメモリ内で扱われるデータ構成とデータ転送について説明します。

また、スタックやキューの概念を述べ、実際の使用について考察しました。

3.1 メモリの構成

MC68000 は、A1～A23 までの 23 本のアドレス出力ピンと、プロセッサ内部の A0 信号により、 2^{24} バイト=16M バイトのアドレス空間を直接アクセスすることができます。

この 16M バイトのアドレス空間は、連続した 8M ワードの構成 (図 3.1 参照) か、あるいは 4M のロング・ワード構成をとることができます。

MC68000 は、他のプロセッサに見られる I/O などの命令はなく、メモリ・マップド I/O 方式 (I/O などのデバイスとのデータ・アクセスにおいて、I/O を特に意識することなく、I/O デバイスをメモリ内に割り付け、見かけ上メモリ to メモリの転送として処理する方式) をとり、ページング・セグメンテーション (アドレス空間を、ある基準に基づいて区分する方法) を用いないために、直接 16M バイトのメモリ構成のどこにでも I/O アドレス領域を複数に割り付けることが可能です。

16M バイトのアドレス空間のうち、16 進 HEX アドレスで 000～0FF までが、システムの例外処理ベクタ・アドレス・テーブルに割り付けられ、また、100～3FF までがユーザの割込みベクタ・アドレス・テーブルに割り当てられています。したがって、ユーザに開放されているアドレスは 000400～FFFFFFF となります (図 3.1 参照)。

MC68000 は、前述した 3 本のファンクション・コード出力信号を四つに分類します。よって、ファンクション・コード出力信号を外部メモリのチップ・イネーブルとして用いることで、最大 64M バイトのメモリ領域をアクセスすることができます。

また、FC2 出力信号を図 3.2 で示すようにチップ・イネーブルとして用いる場合、メモリ・プロテクションをすることが可能です。

3 メモリの構成

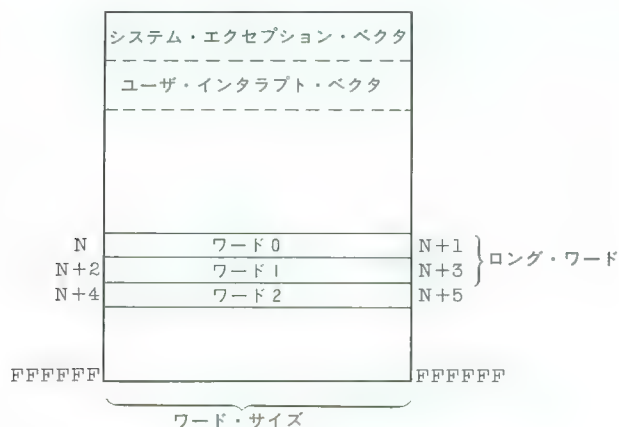


図 3.1 メモリ構成



• EVEN バイト・アドレス・アクセス

AO=0

$\overline{UDS}=0$

$\overline{LDS}=0$

• ODD バイト・アドレス・アクセス

AO=1

$\overline{UDS}=1$

$\overline{LDS}=0$



EVEN
バイト・
アドレス

ODD
バイト・
アドレス

図 3.2 メモリ・マップ

3.2 メモリ内のデータ構成と使用

メモリ空間内のデータは、バイト、ワードおよびロング・ワードのデータ・サイズでアクセスされます。また、命令および複数バイト・データは、ワード単位でのみアクセスされます。メモリ内のデータ構成を図 3.3 に示します。

(1) **バイト単位のデータ構成**：メモリ内をバイト単位のデータ系列として扱う場合、上位バイト（バイト 0）は偶数アドレス N で指定され、下位バイト（バイト 1）は奇数アドレス $N+1$ で指定されます。

(2) **ワード単位のデータ構成**：メモリ内をワード単位のデータ系列として扱う場合、ワード・データは必ず偶数アドレスによりアクセスされます。例えば、ワード 0 のデータはアドレス N により指定され、次のワード・データ；ワード 1 はアドレス $N+2$ により指定されます。

(3) **ロング・ワード単位のデータ構成**：メモリ内をロング・ワード単位のデータ系列として扱う場合、上位ワード（ワード（H））はアドレス N で指定され、下位ワード（ワード（L））はアドレス $N+1$ で指定されます。

MC68000 がバイト単位のデータをアクセスするときのブロック図を図 3.4 に示します。このとき、アドレス・バス $A1 \sim A23$ ，データ・バス $D0 \sim D15$ のほかに、制御ラインとして R/\overline{W} ， \overline{LDS} ， \overline{UDS} ラインを用いてデータ転送を実行します。メモリは、偶数番地メモリと奇数番地メモリに分類され、それぞれ \overline{LDS} ， \overline{UDS} をアクティブ“LOW”のセレクト入力としています。 R/\overline{W} 線は、データ・バス上のデータの方向を決め、 \overline{UDS} 線により、偶数アドレスの $D8 \sim D15$ の 1 バイト転送を可能にし、 \overline{LDS} 線により、奇数アドレスの $D0 \sim D7$ の 1 バイト転送を可能にしています。したがって、プロセッサ内部ではオペコード（命令）を読み取り、オペランド（転送データ）が偶数アドレスのバイト・データ、奇数アドレスのバイト・データ、ワード・データのいずれかを判断して、表 4.1 に示された論理レベルを \overline{UDS} ， \overline{LDS} 線に出力することになります。

3 メモリの構成

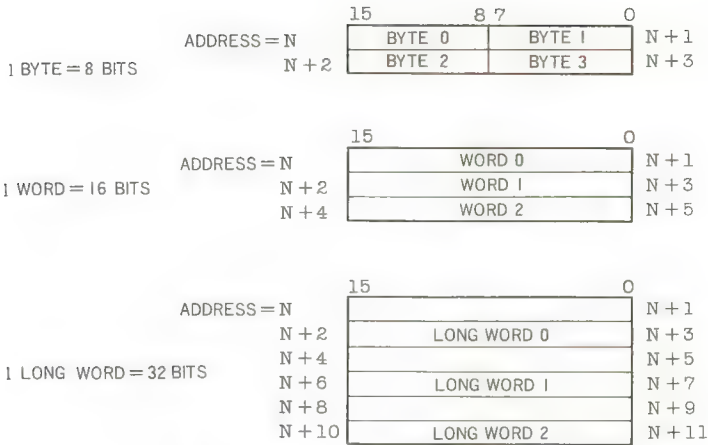


図 3・3 データ・メモリ・フォーマット

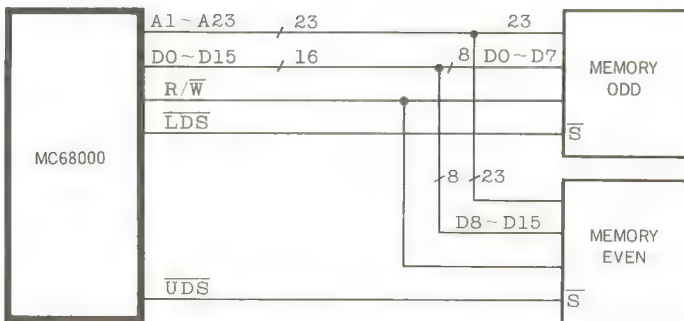


図 3・4 バイト・アドレッシング

3.3 スタックの構成

スタックとは、後入れ先出し（LIFO）の性格を持った配列データ構造のメモリ領域をいいます。MC68000 では、アドレス・レジスタ間接、ポストインクリメント・アドレッシング・モードおよびプリデクリメント・アドレッシング・モードによりスタックをサポートします。データをスタックにセーブすることをプッシュ（push）といい、スタックからデータを引き出すことをプル（pull）といいます。システム・スタックは、例外処理や命令により自動的に使用され、ユーザ・スタックはアドレッシング・モードを用いて作成され操作されます。スタックへのプッシュ・プルは、スタック・ポインタを参照して行います。

〔1〕 **システム・スタック** アドレス・レジスタ A7 が、システム・スタック・ポインタとして使用されます。また、ステータス・レジスタの“S”ビットの状態に応じ、スーパーバイザ・スタック・ポインタ（SSP）あるいはユーザ・スタック・ポインタ（USP）として役目を果たします。すなわち、スーパーバイザ・モードでは SSP としてのみ働き、ユーザ・モードでは USP としてのみ働くことを示し、一方のモードにおいて双方のスタック・ポインタのアクセスを防いでいます。

トラップや割り込みなどの例外処理において、アドレス・レジスタ A7 はスーパーバイザ・スタック領域へセーブされます。

ユーザ・モードで用いられる RTS 命令などにより、A7 はユーザ・スタック・ポインタとして自由に使用できます。スタック操作は一般にワード単位で行い、システム・スタック内のデータ配列をワード構成に保っています。このため、バイト・データのスタック操作では、上位半分が使用され、下位半分は使用されません。

〔2〕 **ユーザ・スタック** システム・スタックが、例外処理や RTS 命令などにより自動的にスタック操作を行うのに対し、ユーザ・スタックは、アドレス・レジスタ（A0～A6）をスタック・ポインタとして利用し、ポストインクリメントおよびプリデクリメントのアドレス・レジスタ間接アドレッシング・モードを

使用することにより作成され操作されます。

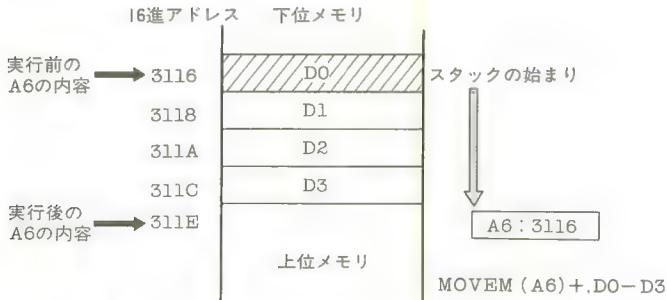


図 3・5 ポストインクリメントを用いたプッシュ操作
プッシュ

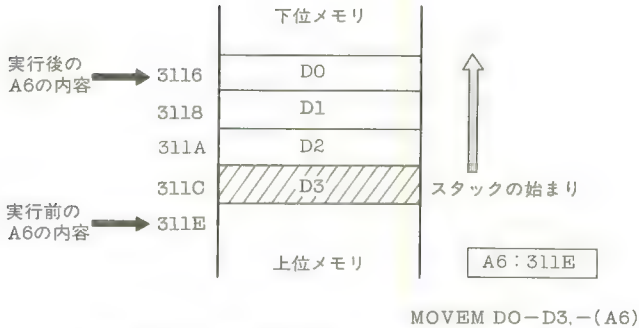


図 3・6 プリデクリメントを用いたプッシュ操作

3.4 キューの構成

キューとは、先入れ先出し（FIFO）で用いられる配列データ構造のメモリをいいます。MC68000 では、ポストインクリメントまたはプリデクリメント・アドレス・レジスタ間接のアドレッシング・モードを利用することにより、ユーザのキューを作成し操作することができます。

キューは、データを先端から取り入れ、終端から取出しを行うために2個のアドレス・レジスタ（A0～A6 のうち2個）を使用します。終端を指すアドレス・レジスタを**プット・ポインタ**と呼び、先端を指すアドレス・レジスタを**ゲット・ポインタ**と呼びます。

また、プット・ポインタのアドレス更新とともにゲット・ポインタの指すアドレスも更新され、常にポインタ間のバッファ容量を一定に保ちます。

このようにして、ユーザは、スーパーバイザ・メモリ・アクセスにおけるユーザ保護を実現します。

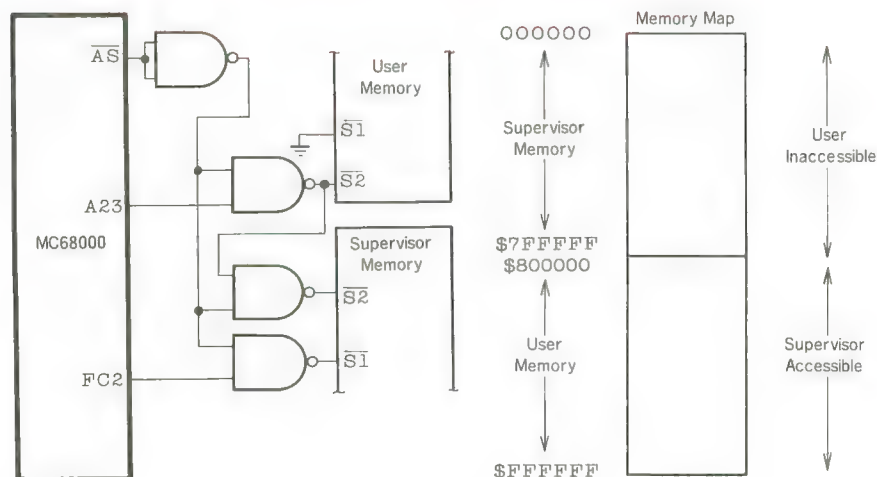


図 3.7 スーパーバイザ・メモリ・アクセスにおけるユーザ保護

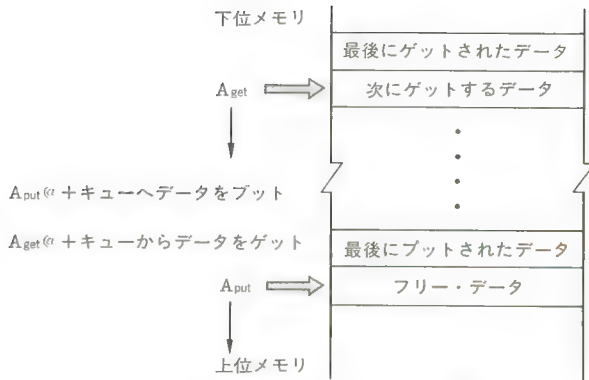


図 3・8 アドレスの上位へ向うキュー操作

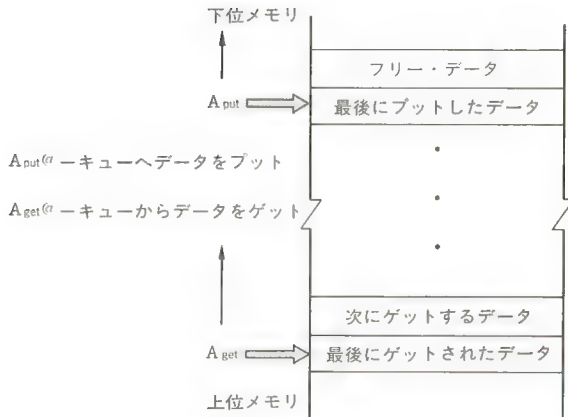


図 3・9 アドレスの下位へ向かうキュー操作

3 メモリの構成

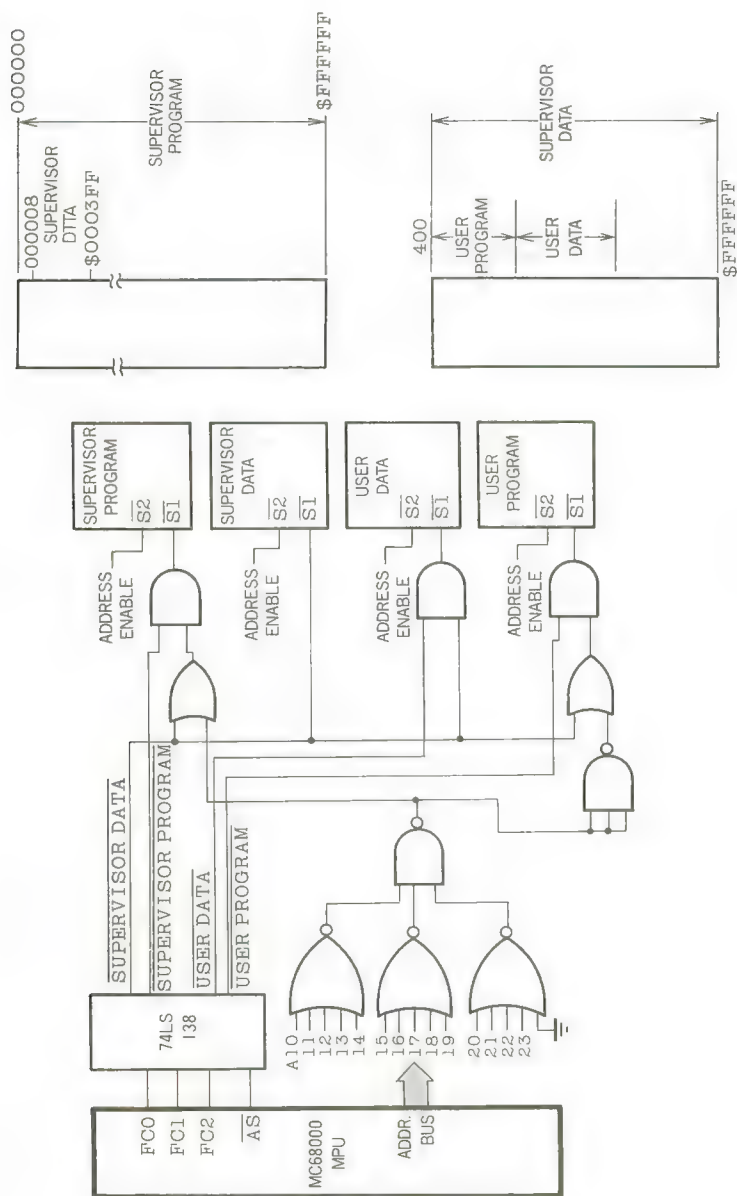


図 3・10 デュアル・メモリ・マップ

4. 入/出力の構成

68000 全体の入出力信号を図示し，入出力に関するシグナル名と，その機能について説明しています。データ・バスとデータ・ストローブの関係や，何種類かのバス・サイクルとシグナルの関係，およびデータ転送の関係も図示しています。

4.1 バス・シグナル

入/出力におけるバス・シグナルは、機能別にアドレス・バスとデータ・バスの二つの機能に分けられます。アドレス・バスとデータ・バスは、それぞれ独立した並列バスであり、非同期動作と MC 6800 系の同期式でデータを転送（送/受信）します。またどのようなバス・サイクルでも、バス・マスタは、すべての信号動作をそのバス・サイクル内で完了させなくてはなりません。

68000MPU と外部デバイス間のデータは、16 本のデータ・バス（D0～D15）を通して行われます。したがって、メモリや入/出力間でやりとりできるデータは最大 16 ビットです。また、16 ビットより大きいデータを転送するためには、2 回以上の転送が必要となります。

[1] データ・バス

- D0 から D15 のデータ・バスは、16 ビットの双方向性 3 ステート・バスです。
- バイト長やワード長のデータ転送（送/受信）を行います。
- 割込みアクノリッジ・サイクル中は、外部チップからデータ・ライン D0～D7 にベクタ番号が入ります。

[2] アドレス・バス

- A1 から A23 のアドレス・バスは、23 ビットの単方向性 3 ステート・バスです。
- 16M バイト（16 777 216 バイト）または 8M ワード（8 388 600 ワード）のデータを直接アドレス可能です。
- アドレス・ストローブライン（ \overline{AS} ）をアサートすることによりアドレス・バスに意味のあるアドレスを示します。
- 割込みサイクルを除いたすべてのサイクルにおいて、バス・オペレーション用のアドレスを決定します（6.1 節の割込み参照）。

4 入 / 出 力 の 構 成

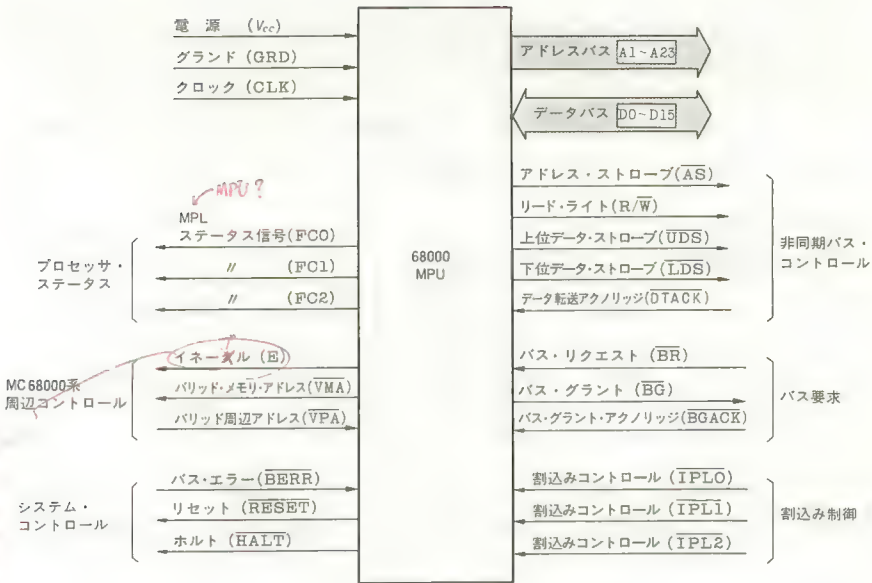


図 4・1 MC68000 の入/出力信号

表 4・1 データ・バスのデータ・ストロブ・コントロール

UDS	LDS	R/W	D8~D15	D0~D7
H	H	-	(意味なしデータ)	(意味なしデータ)
L	L	H	データ・ビット 8~15	データ・ビット 0~7
H	L	H	(意味なしデータ)	データ・ビット 0~7
L	H	H	データ・ビット 8~15	(意味なしデータ)
L	L	L	データ・ビット 8~15	データ・ビット 0~7
H	L	L	データ・ビット 0~7	データ・ビット 0~7
L	M	L	データ・ビット 8~15	(データ・ビット 8~15)

Read
Write

8~15 ではないか?
0~7 ではないか?

CPUからはデータが出て行く
メモリ (or バイナリ) には
受け取らない、ということかな?

4.2 バス・コントロール・シグナル

バス・コントロール・シグナルは、二つの機能に大別でき、それぞれ5種類の非同期式バス・コントロール信号と3種類のバス・アービトレーション・コントロール信号があります。

[1] 非同期式バス・コントロール信号

\overline{AS} (アドレス・ストローブ) : この出力信号がアクティブのときは、アドレス・バス上に有効なアドレスが存在していることを示します。

R/\overline{W} (リード/ライト) : この出力信号は、データ・バスの転送がリード・サイクル (H) かライト・サイクル (L) かを示します。

\overline{UDS} , \overline{LDS} (上位/下位データ・ストローブ) : この出力信号は、データ・バス上のデータのバイト単位やワード単位を指定し、 R/\overline{W} シグナルとともに動作します。

\overline{DTACK} (データ転送アクノリッジ) : この入力信号は、データ転送が完了していることを示します。また、MPU がリード・サイクルやライト・サイクル中に \overline{DTACK} を認知すると、データをラッチしてバス・サイクルを終了させます。

[2] バス・アービトレーション・コントロール信号

\overline{BR} (バス・リクエスト) : この入力信号は、他のデバイスがバス・マスタになることを MPU に要求していることを示し、すべてのバス・マスタとなりうるデバイスをワイヤード OR にします。

\overline{BG} (バス・グラント) : この出力信号は、MPU のバス・サイクルの終了した時点でバス制御を解除し、他のデバイスのバス・マスタの要求を移すことを示します。

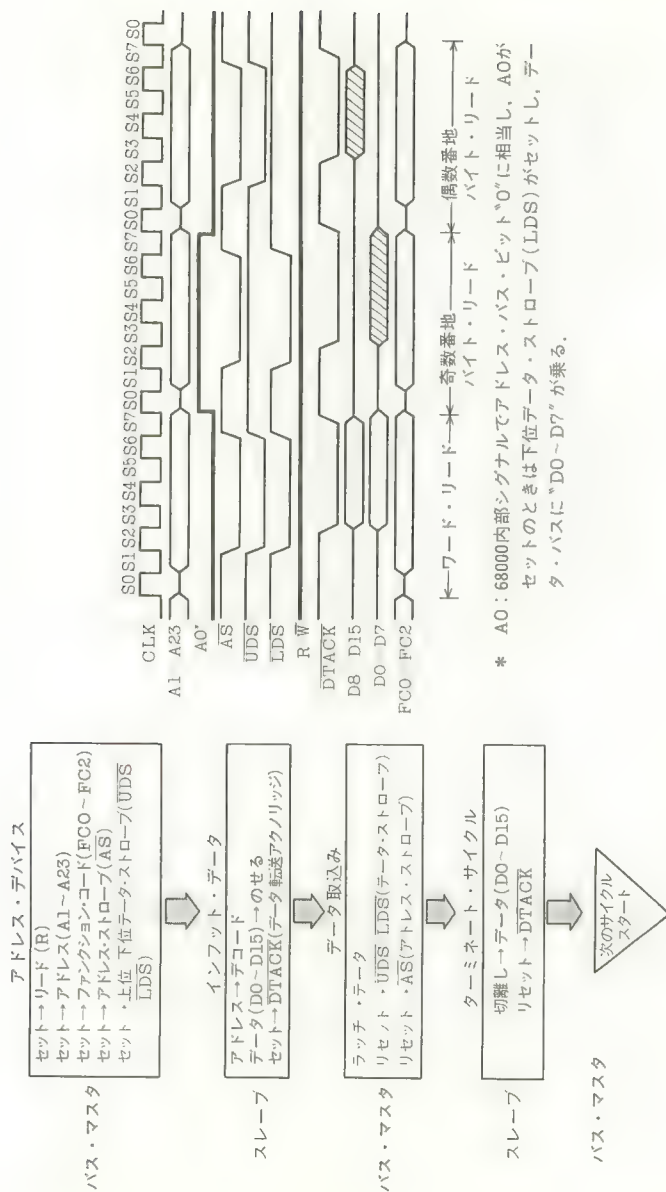


図 4.2 ワード・リード・サイクル・フローとタイミング

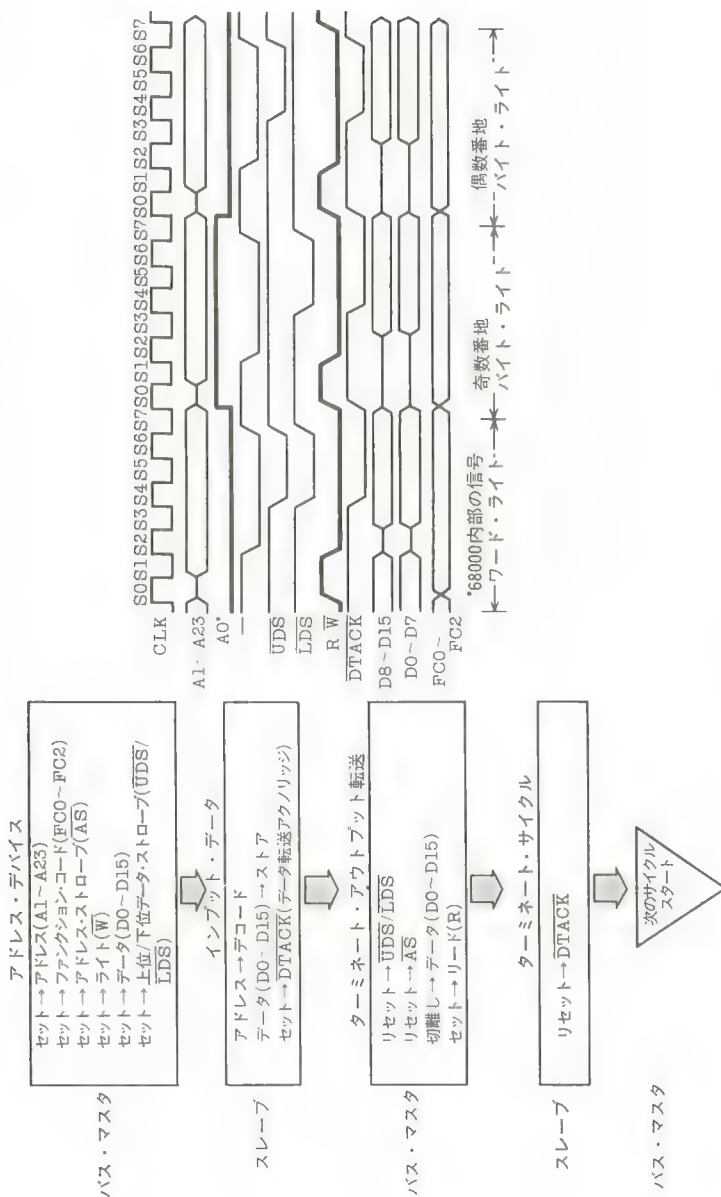


図 4.3 ワード・ライト・サイクル・フローとタイミング

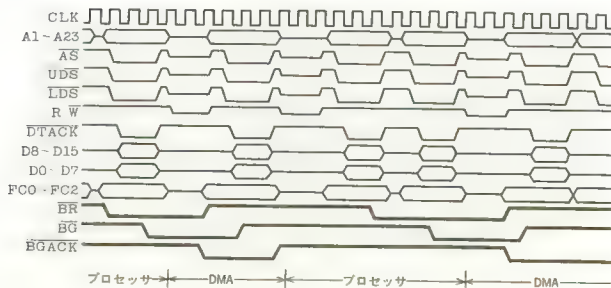
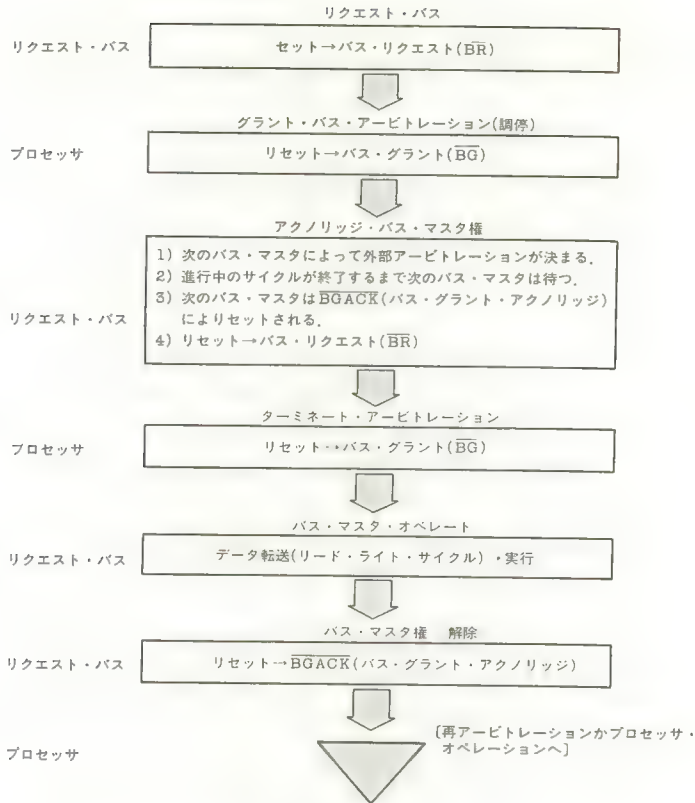


図 4.4 バス・アービトレーション・サイクル・フローとタイミング

4 入/出力の構成

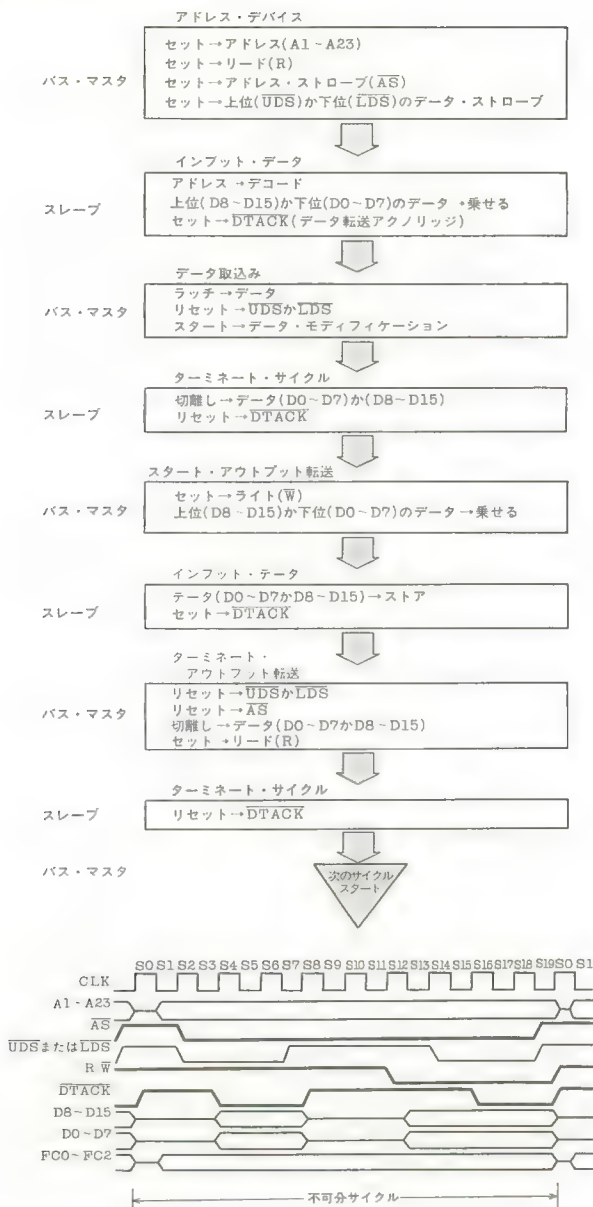


図 4.5 リード・モディファイ・ライト・サイクル・フローとタイミング

〔3〕 バス・アービトレーション

バス・アービトレーションは、プロセッサと外部デバイス間でのやりとりに、バス要求 (\overline{BR})、バス・グラント (\overline{BG})、バス応答 (\overline{BGACK}) の三つのシグナルにて制御されています。

バス・アービトレーション・ユニットの状態図を図4・6に示します。

a. バス要求 (\overline{BR}) バス・マスタになりうる外部デバイスは、バス要求信号 (\overline{BR}) をアサートすることによって、プロセッサに外部デバイスが外部バスのコントロールを要求していることを知らせます。

b. バス・グラント (\overline{BG}) バス要求がアサートされることにより、プロセッサは、可能な限り速やかにバス・グラント (\overline{BG}) をアサートし、バス・アービトレーション承認をします。

c. バス・マスタの応答 (\overline{BGACK}) バス・グラントがアサートされると要求を出した外部デバイスは、バスマスタ応答 (\overline{BGACK}) をアサートし新しいバスマスタとなります。

プロセッサ・バス・サイクルにおけるバス・インアクティブどきの特殊例としておのおののバス・アービトレーションについて説明します。

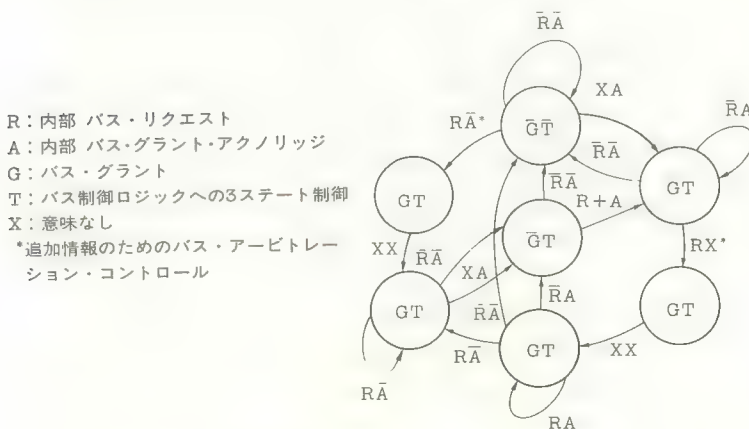


図 4・6 MC68000 バス・アービトレーション・ユニット図

4 入/出力の構成

図4・7は、プロセッサのバス・サイクル内でのバス・アービトレーションのタイミング図で、バス要求デバイス信号をアサートし3ステートからのバスの解放とプロセッサの次のバス・サイクルまでを示します。

図4・8は、バスがインアクティブなときのバス・アービトレーションのタイミング図で、乗算や割算命令などの内部動作実行時のシーケンスを示します。

図4・9は、プロセッサのバス・サイクルの特別な場合のバス・アービトレーションのタイミング図で、MPUがバス・サイクルを開始したが、まだアドレス・ストローブ信号(\overline{AS})がアサートされていないときにバス要求が起きてもバスグラントは次の立ち上がりではアサートされず、内部でのアサートから二つ目の立ち上がりでバスグラントがアサートされます。

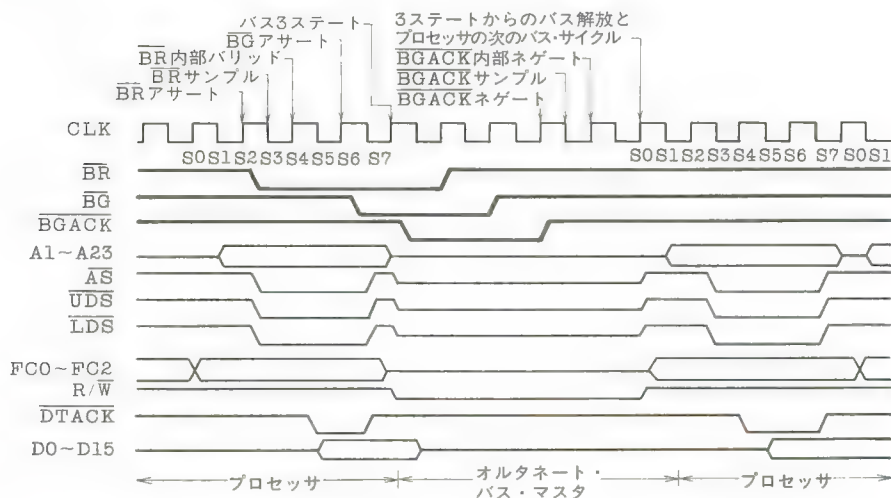


図 4・7 プロセッサ・バス・サイクルにおけるバス・アービトレーション

4 入/出力の構成

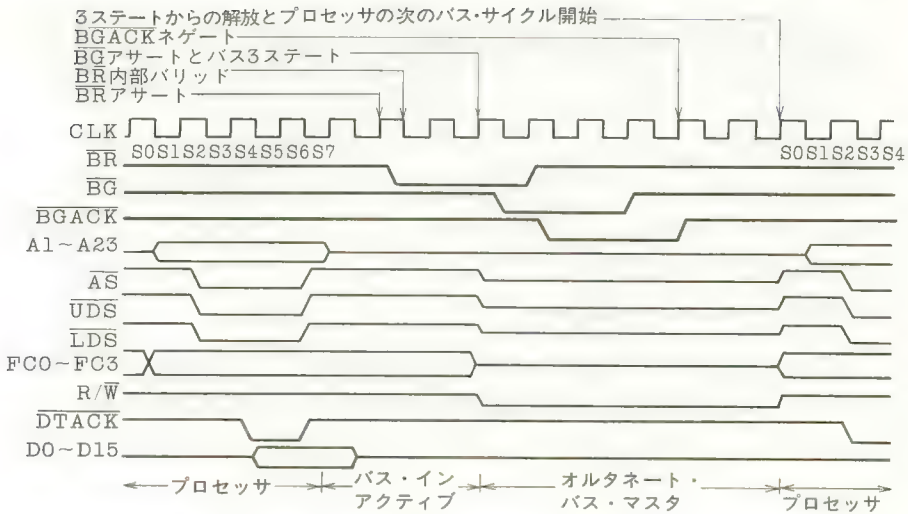


図 4・8 バス・インアクティブにおけるバス・アービトレーション

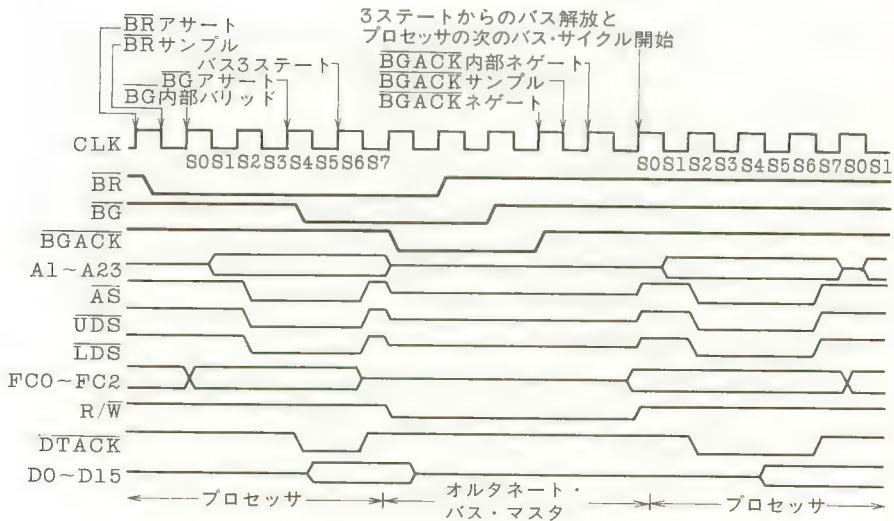


図 4・9 特殊例のプロセッサ・バス・サイクルにおけるバス・アービトレーション

4.3 MPU コントロール・シグナル

MPU コントロール・シグナルは割込みコントロール、システム・コントロール、68 系コントロールの 3 種類の機能に分けられ、それぞれが 3 種類の割込みコントロール信号、3 種類のシステム・コントロール信号と、3 種類の MC6800 系コントロール信号があります。

〔1〕 割込みコントロール信号

$\overline{\text{IPL0}}$, $\overline{\text{IPL1}}$, $\overline{\text{IPL2}}$: 3 本の入力信号は、外部から割込みを要求するデバイスの優先レベルを決めるものです。 $\overline{\text{IPL0}}$ が最下位ビット、 $\overline{\text{IPL2}}$ が最上位ビットで、レベル“1”がもっと低位で、レベル“7”が最優先レベルを示します（図 2-5 参照）。
最も

〔2〕 システム・コントロール信号

$\overline{\text{BERR}}$ (バス・エラー): この入力信号は、実行中のバス・サイクルに問題が生じたことを MPU に知らせます。

$\overline{\text{RESET}}$ (リセット): この双方向信号は、外部からのリセット信号で MPU をリセットしたり、RESET 命令で MPU で作成されたりリセット信号で他の外部デバイスをリセットします。

$\overline{\text{HALT}}$ (ホルト): この双方向信号は、外部からホルト信号を受け取ると、MPU は動作中のバス・サイクルを完了した時点で停止します。また、MPU がダブル・バス・ホルト状態で命令実行を停止したときは、MPU はホルト信号を出し、外部デバイスに対して MPU が停止したことを示します。

〔3〕 68 系コントロール信号

E (イネーグル): この出力信号は、MC6800 系の周辺デバイスが使用する同期クロック信号です。

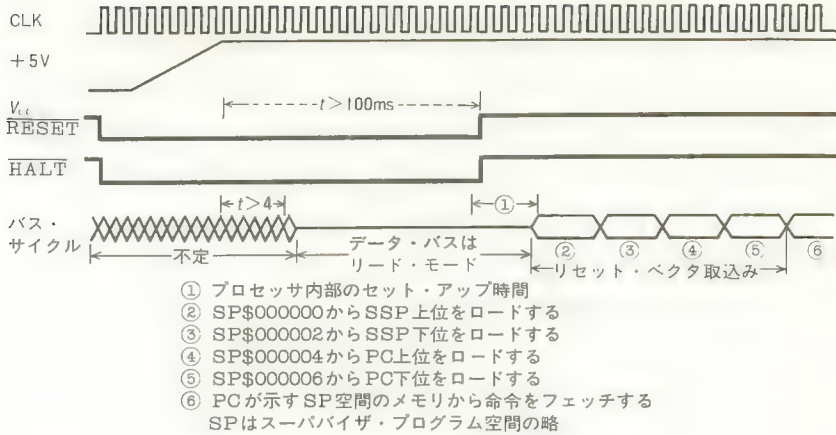


図 4・10 リセット・オペレーションのタイミング

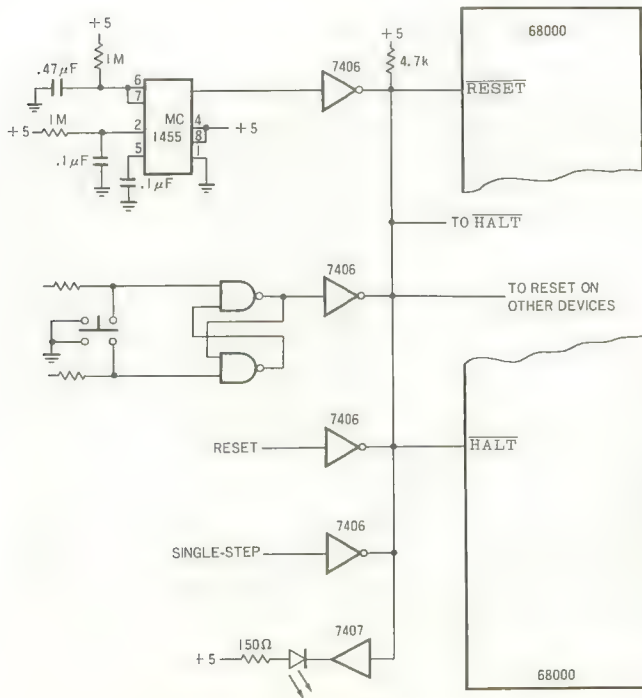


図 4・11 リセットとホルト回路

4 入/出力の構成

VPA (バリッド周辺アドレス)：この入力信号は、MPU がアクセスするデバイスと MC6800 系デバイスのアドレスです。また、外部割込みアクノリッジ・サイクルで、MPU の割込み処理にあたって自動ベクタ動作を行うよう指示します。

VMA (バリッド・メモリ・アドレス): この出力信号は, MC6800 系周辺デバイスに対して, アドレス・バスに有効アドレスがあることを示します. また, MPU がイネーブルと同期していることを示します.

MC68000 のホルト入力信号は、ホルト (HALT)、ラン (RUN)、シングル・ステップ (SINGLE-STEP) 機能を備え、1) アドレス・ライン、2) データ・ライン、3) ファンクション・コードなどの **3 ステート信号** をハイ・インピーダンスにします。なお、MPU がホルトを受け付けている間は、バス・アービトレーションは通常と同じ動作をします。

ホルト機能とハードウェアによる機能を用いることにより、ハードウェア・デ

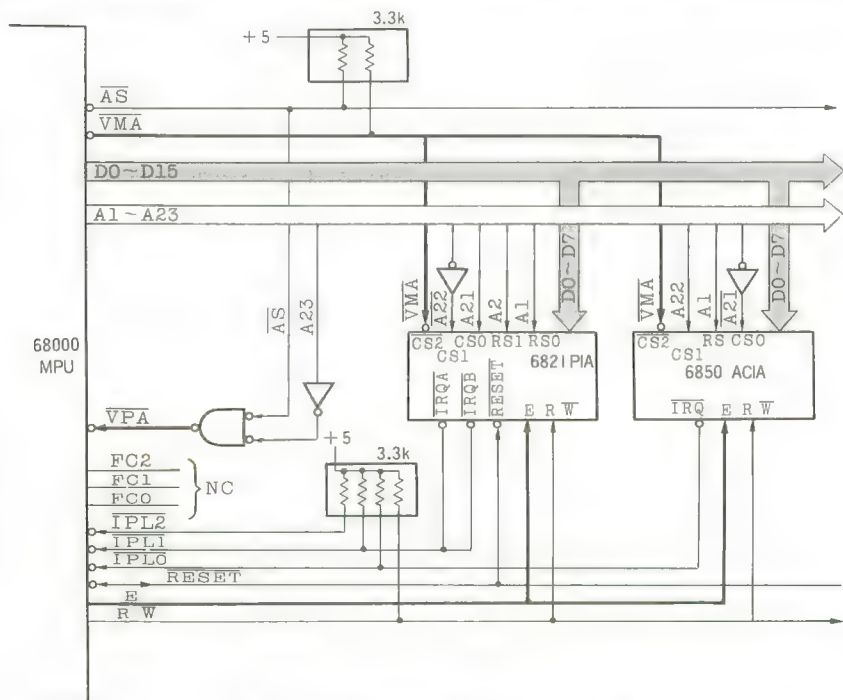


図 4・12 MC6800 系のインタフェース例

4 入 / 出 力 の 構 成

バグガ（シングル・ステップ・モード）でプロセッサを1バス・サイクルまたは1命令ずつトレースすることにより、これらの機能やソフトウェア・デバッグ機能などフレキシビリティのある高いデバッグ能力を備えています。

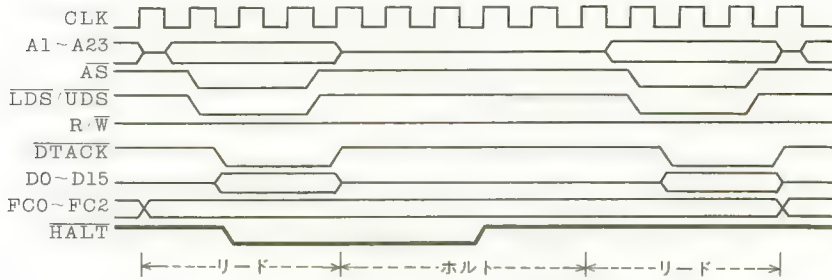


図 4・13 バス・エラーでないときのホルト・タイミング

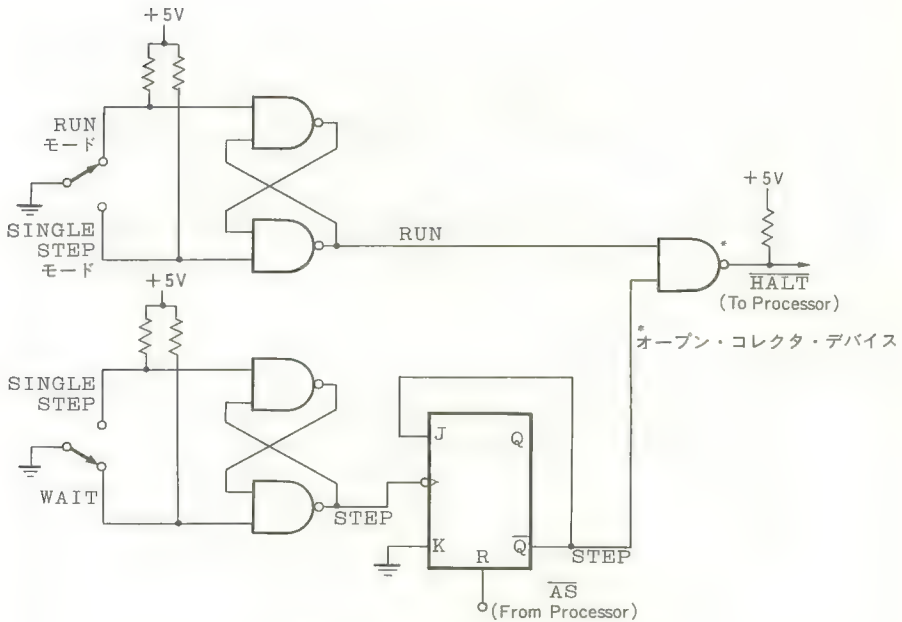


図 4・14 シングル・ステップ回路

4.4 MPU ステータス・シグナルとクロック

プロセッサのメモリ・アクセス状態と割り込み受けを示す3種のステータス信号、別名ファンクション・コードともいい、外部デバイスに対して68000の動作状況を伝えます。

〔1〕 プロセッサのステータス信号

FC0, FC1, FC2: 3本の出力信号は、ユーザ・データか、ユーザ・プログラムか、スーパーバイザ・データか、スーパーバイザ・プログラムか、または割り込みアクノリッジか、のいずれを指定しているのかを外部デバイスに示します。また、**FC0~FC2** 信号で示される情報は、 \overline{AS} シグナル（アドレス・ストローブ）がアクティブであれば常に有効です（表4.2参照）。

〔2〕 クロック入力

CLK: クロック入力は、TTL とコンパチブルな信号であり、一定周波数の信号（50% のデューティ・サイクルが望ましいですが）であれば、特別なクロック・ドライバを用意することなく用いることができます。クロック・パルス回路例を図4.15に示します。

68000 では、4MHz (68000L4)、6MHz (68000L6)、8MHz (68000L8)、10MHz (68000L10)、12MHz (68000L12) の各バージョンがあります。

〔備考〕 68×××L×; セラミック・パッケージ

68×××P×; プラスチック・パッケージ

68×××ZB×; チップキャリア・パッケージ

4 入 / 出 力 の 構 成

表 4・2 ファンクション・コードと機能の関係

FC2	FC1	FC0	サイクル・タイプ (機能)
0	0	0	—
0	0	1	ユーザ・データ
0	1	0	ユーザ・プログラム
0	1	1	—
1	0	0	—
1	0	1	スーパーバイザ・データ
1	1	0	スーパーバイザ・プログラム
1	1	1	割込みアクノリッジ

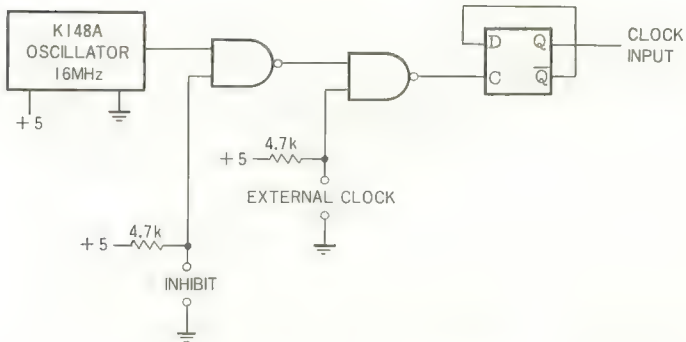


図 4・15 クロック・パルス供給回路

4.5 データ転送

8ビット MPU6800 は、同期式のデバイスとしかデータの送受信ができませんでしたが、16ビット MPU68000 では、同期式でも非同期式でも両方式のデバイスとインタフェースが可能です。

つまり、アクセス・タイムの遅いものには、それなりにゆっくりと、アクセス・タイムの速いものには、それなりに速く合わせるというように、実行速度を順応させることができます。

68000 は、データ転送において通常のリード・サイクルとライト・サイクル、マルチ・プロセッサ構成時のリード・モディファイ・ライト・サイクル、6800 周辺サイクルなどがあります。

リード・サイクル：リード・サイクルは、プロセッサがメモリや周辺デバイスから複数のバイト・データを受け取るサイクルです。1) ワード・リード・サイクル、2) バイト・リード・サイクル、3) ワード・バイト・サイズ・リード・サイクルなどがあります。

A0 ビット “0” は上位データ・ストロブ ($\overline{\text{UDS}}$) を出力。

A0 ビット “1” は下位データ・ストロブ ($\overline{\text{LDS}}$) を出力。

ライト・サイクル：ライト・サイクルの時間内で、プロセッサはメモリや周辺デバイスに複数のバイト・データを送り出すサイクルです。1) ワード・ライト・サイクル、2) バイト・ライト・サイクル、3) リード・モディファイ・ライト・サイクルなどがあります。

A0 ビット “0” は上位データ・ストロブ ($\overline{\text{UDS}}$) を出力。

A0 ビット “1” は下位データ・ストロブ ($\overline{\text{LDS}}$) を出力。

〔1〕 **非同期式バス・コントロール** データ転送のコントロールは、2本の制御シグナルである $\overline{\text{UDS}}$ と $\overline{\text{LDS}}$ によって、バイト単位やワード単位のデータを指定します。

通常のリード・サイクルは **4 クロック・サイクル** が必要であり、ライト・サイクルのときは **5 クロック・サイクル** が必要です。

4 入 / 出力 の 構 成

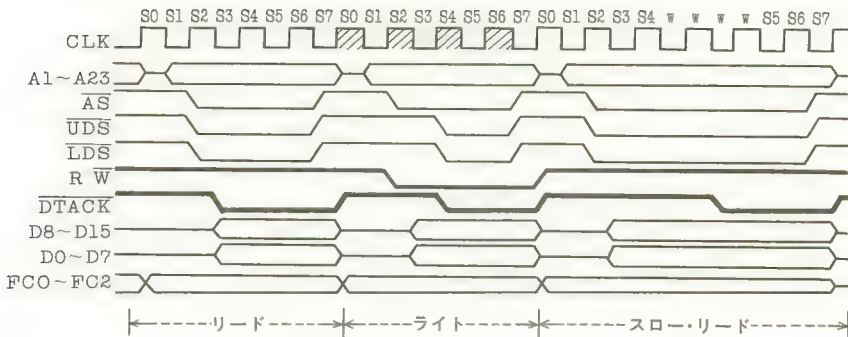


図 4・16 リード・サイクルとライト・サイクルのタイミング

〔2〕 同期式バス・コントロール MC6800 系のバスサイクルは、6800 系周辺デバイス（PIA や ACIA）が使用できるように、同期式バス・サイクルを非同期式の 68000 バス・サイクルに変更させます。インタフェースするための同期式バス制御ラインは、 \overline{E} と非同期式バス信号 \overline{DTACK} に代わる信号として \overline{VPA} ，そして \overline{VMA} の 3 種類のシグナルです（図 2・5，図 4・12 参照）。

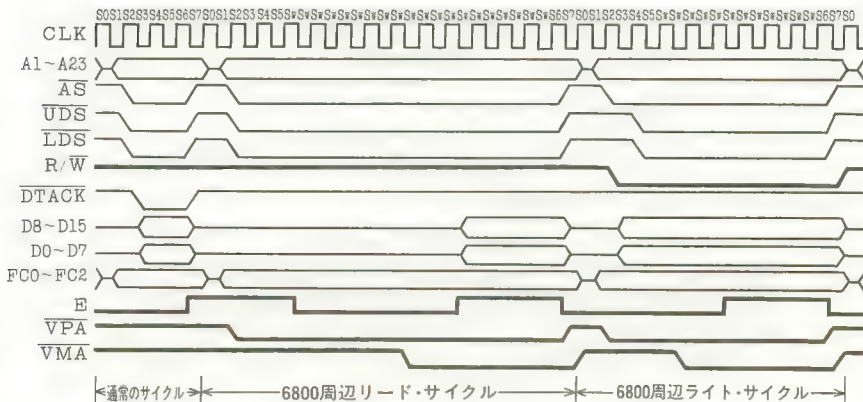


図 4・17 6800 系周辺バス・サイクルのタイミング

4 入 / 出 力 の 構 成

クロックの見方

S_0 (ステート 0) : バス・サイクル開始時の 1 番目のクロックが “H” の時間

S_1 (ステート 1) : バス・サイクル開始時の 1 番目のクロックが “L” の時間

S_2

\vdots




S_n (ステート n) : n 番目のクロック

\vdots

S_W (ステート W) : デバイスからの応答をプロセッサが待つ時間

クロックとバス・サイクルの関係

クロックとバス・サイクルの関係

クロック 周 波 数	周 期	リード・ サイクル	ライト・ サイクル
4 MHz	250ns 	1000ns	1250ns
8 MHz	125ns 	500ns	625ns
10 MHz	100ns 	400ns	500ns

5. アドレッシング・モード

68000 の特長であるアドレッシング・モードは、モードを多数備え、アドレス計算や転送命令の省略化による効率良いプログラミングの作成と速度の向上を提供します。

7 種類のアドレス・モードを各モード単位で具体例にて図示し、配列データ処理法、プログラムの再配置法、レジスタの自動的な増減法、メモリ・スタックの作成法などを解説しています。

5.1 実効アドレス

各命令のオペランドのバリエーションを決定するのに必要な要素として、アドレッシング・モードがあげられます。68000 のアドレッシング・モードは六つの基本グループに分けられ、総計 14 種類のアドレッシング・モードを備え、その種類の豊富さと柔軟さによって、プログラミングのステップ数の省略や効率の向上、実効速度の向上ができます。また、繁雑になりがちなものも整然とした管理につながっていきましょう。

14 種類のアドレッシング・モードは、2 種のレジスタ直接、5 種類のレジスタ間接、2 種類のアブソリュート、2 種類のプログラム・カウンタ相対、2 種類のイミディエート、1 種類のインプライドを備えています（表 5.1 参照）。

ほとんどの命令では、オペランドの位置はオペレーション・ワードの実行アドレス・フィールドを使用します。実行アドレス・フィールドは、レジスタ・フィールド（ビット 0～2）とモード・フィールド（ビット 3～5）からなっています。レジスタ・フィールドはレジスタ番号を指定し、モード・フィールドは、いろいろなモードを選択します（表 5.2 参照）。

15					10						5		2	0
X	X	X	X	X	X	X	X	X	X	X	実効アドレス			
											モード		レジスタ	

図 5.1 単一実効アドレス命令のオペレーション・ワードの一般的な型

5 アドレッシング・モード

表 5・1 基本グループとアドレッシング・モード

基本型	モード	機能
レジスタ直接	1) データ・レジスタ直接 2) アドレス・レジスタ直接	$EA = D_n$ $EA = A_n$
レジスタ間接	3) アドレス・レジスタ間接 4) ポストインクリメント 5) プリデクリメント 6) オフセット付き 7) インデックスおよびオフセット付き	$EA = (A_n)$ $EA = (A_n), A_n \leftarrow A_n + N$ $A_n \leftarrow A_n - N, EA = (A_n)$ $EA = (A_n) + d_{16}$ $EA = (A_n) + (X_n) + d_8$
アブソリュート (絶対)	8) アブソリュート・ショート 9) アブソリュート・ロング	$EA = (\text{次のワード})$ $EA = (\text{次の2ワード})$
プログラム・カウンタ 相対	10) オフセット付き 11) インデックスおよびオフセット付き	$EA = (PC) + d_{16}$ $EA = (PC) + (X_n) + d_8$
イミディエート	12) イミディエート 13) クイック・イミディエート	DATA = 次のロード Inherent DATA
インプライド(合意)	14) インプライド・レジスタ	$EA = SR, USP, SP, PC$

EA=実効アドレス

A_n =アドレス・レジスタ

D_n =データ・レジスタ

SR=ステータス・レジスタ

PC=プログラム・カウンタ

X_n =インデックス・レジスタとして用いられるアドレス・レジスタまたはデータ・レジスタ

()=内容

←=転送

d_8 =8ビット・オフセット

d_{16} =16ビット・オフセット

$N=1$ (バイトのとき)

2(ワードのとき)

4(ロング・ワードのとき)

表 5・2 アドレッシング・モードのコード別コード表

アドレス・モード	モード	レジスタ
データ・レジスタ直接	000	レジスタ番号
アドレス・レジスタ直接	001	レジスタ番号
アドレス・レジスタ間接	010	レジスタ番号
ポストインクリメント・アドレス・レジスタ間接	011	レジスタ番号
プリデクリメント・アドレス・レジスタ間接	100	レジスタ番号
ディスプレイースメント・アドレス・レジスタ直接	101	レジスタ番号
インデックス・アドレス・レジスタ間接	110	レジスタ番号
アブソリュート(絶対)・ショート	111	000
アブソリュート(絶対)・ロング	111	001
ディスプレイースメント・プログラム・カウンタ	111	010
インデックス・プログラム・カウンタ	111	011
イミディエート/ステータス・レジスタ	111	100

5.2 レジスタ直接モード

レジスタ直接モードには、データ・レジスタ直接とアドレス・レジスタ直接の2種類があります。実効アドレッシングは、8個のデータ・レジスタ(D0～D7)または8個のアドレス・レジスタ(A0～A7)の中のいずれか1個をオペランドとして指定できます。

[1] **データ・レジスタ直接** オペランド・データは、実効アドレス・レジスタ・フィールドで指定したデータ・レジスタに格納されます。

機能: $EA = Dn$

PTL 表記: Dn

モード: 000

レジスタ: n

データ・レジスタ Dn : オペランド

例 `MOVE D0, $1000`

[2] **アドレス・レジスタ直接** オペランド・データは、実効アドレス・レジスタ・フィールドで指定したアドレス・レジスタに格納されます。

機能: $EA = An$

PTL 表記: An

モード: 001

レジスタ: n

アドレス・レジスタ An : オペランド

例 `MOVE A4, $202000`

5 アドレッシング・モード

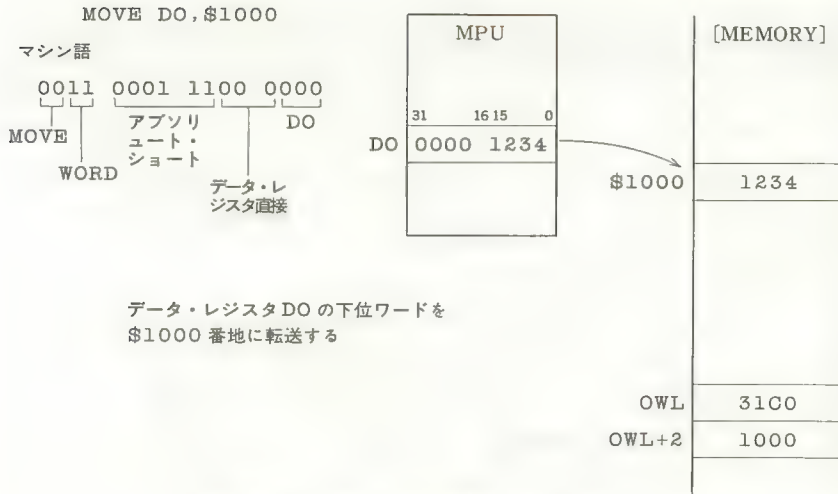


図 5・2 データ・レジスタ直接アドレッシング

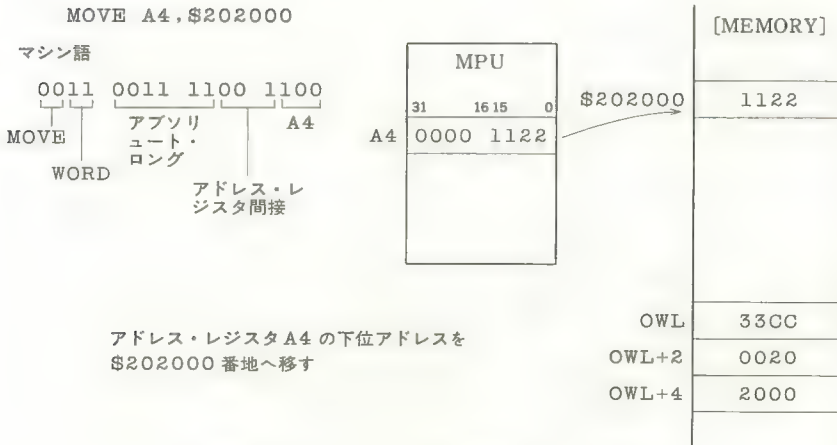


図 5・3 アドレス・レジスタ直接アドレッシング

5.3 レジスタ間接モード

レジスタ間接アドレッシング・モードは、データはメモリに格納されており、オペランドでそのデータのアドレスを指定します。レジスタ間接モードとして次の5種類があり、どのモードを用いるかによって実効アドレスとアドレス・レジスタの関係が決定されます。

1) アドレス・レジスタ間接, 2) ポストインクリメント・アドレス・レジスタ間接, 3) プリデクリメント・アドレス・レジスタ間接, 4) オフセット付きアドレス・レジスタ間接, 5) インデックス・アドレス・レジスタ間接

〔1〕 **アドレス・レジスタ間接** 5種類のモード中で最もシンプルな型であるアドレス・レジスタ間接モードは、オペランド・データを示すアドレス・レジスタ・フィールドで指定されるアドレス・レジスタに格納され、ジャンプ命令(JMP)やジャンプ・サブルーチン命令(JSR)を除いて、オペランドによってデータ位置を示します。

機能: $EA = (An)$

PTL表記: $An@$

モード: 010

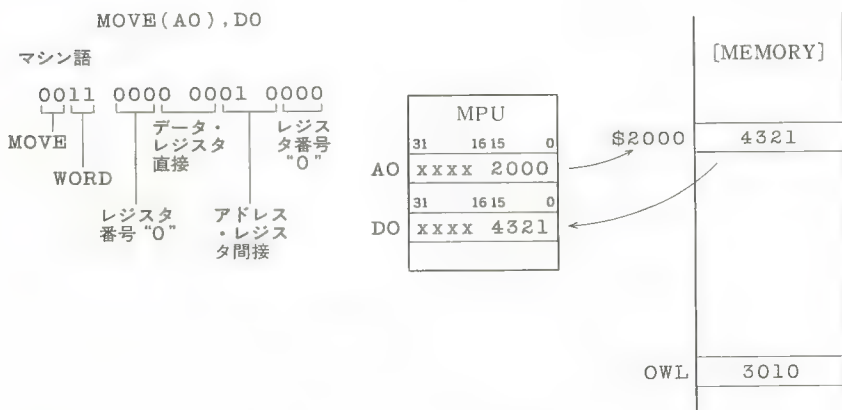


図 5.4 アドレス・レジスタ間接

PALIN:1 MC68000 ASM

```

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23

*      DETECTING A 16-BIT PALINDROME
*
** A 16-BIT WORD POINTED TO BY A0
** IS CHECKED FOR SYMMETRY, I.E.,
** THE WORD READS THE SAME LEFT
** TO RIGHT AS IT READS RIGHT TO
** LEFT. IF IT IS DETERMINED TO BE
** A PALINDROME, THE BYTE TO WHICH
** THE STACK POINTER IS POINTING
** AFTER RTS WILL BE ALL 1'S
*
      ORG $1000
      PALCHK MOVEM.L D0/D1/D2, -(SP)
      MOVEQ #7, D2      * 7 --> COUNTER
      MOVE (A0), D0     * TEST WORD --> D0.W
      AGAIN LSR 1, D0    * LSB, D0.W --> X,CCR
      ROXL 1, D1         * X,CCR --> LSB, D1.W
      DBRA D2, AGAIN    * DO THIS LOOP 8 TIMES
      CMP.B D0, D1      * IF MSB OF TST WRD=MIRROR IMAGE OF LSB
      SEQ 16(SP)        *THN $FF --> 16(SP),ELS 00 --> 16(SP)
      MOVEM.L (SP)+, D0/D1/D2
      RTS
      END

```

***** TOTAL ERRORS 0-- 0

SYMBOL TABLE

AGAIN	001008	PALCHK	001000
-------	--------	--------	--------

PALINDROM チェック・プログラム例

5 アドレッシング・モード

レジスタ：n

アドレス・レジスタ An :

メモリ・アドレス



メモリ・アドレス :

オペランド

例 MOVE(A0), D0

ポストインクリメントとプリデクリメントは、実行した後にアドレス・レジスタの値を増したり減らしたり自動的に更新し、メモリのデータを他の場所に、増加するアドレス順や減少するアドレス順に転送するのに有効です。

〔2〕 **ポストインクリメント** オペランド・データを示すアドレスは、レジスタ・フィールドで指定されるアドレス・レジスタに格納されます。命令を実行した後に指定されたアドレス・レジスタでバイト、ワード、ロング・ワードのデータ・サイズの指定に応じて、指定されたアドレス・レジスタを1（バイト）、2（ワード）、4（ロング・ワード）を加算します。

機能： $An = An + N$, $EA = (An)$

RTL 表記：An@+

モード：011

レジスタ：n

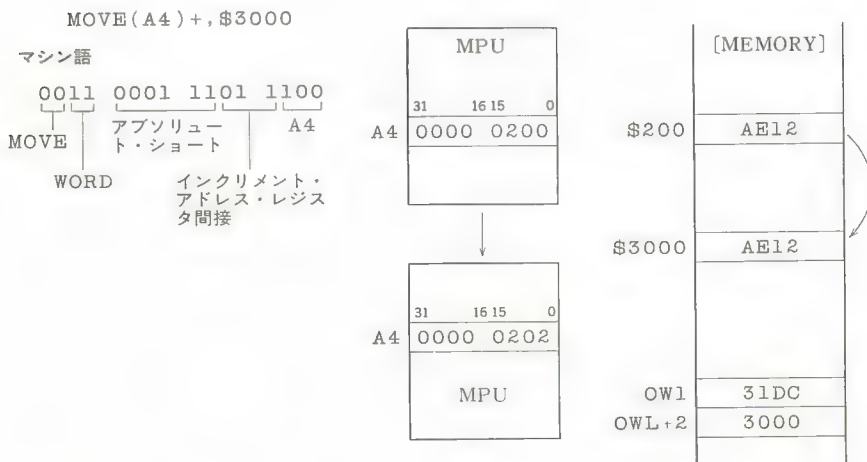


図 5・5 ポストインクリメント付きアドレス・レジスタ間接

アドレス・レジスタ

① オペランド長さ 1, 2, 4

例 MOVE (A4)+, \$3000

[3] プリデクリメント オペランドのアドレスは、レジスタ・フィールドで指定されるアドレス・レジスタに格納され、命令を実行する前に指定されたアドレス・レジスタで、バイト、ワード、ロング・ワードのデータ・サイズの指定に応じて、指定されたアドレス・レジスタを1（バイト）、2（ワード）、4（ロング・ワード）を減算します。

機能： $A_n = A_n - N$, $EA = (A_n)$ RTL 表記： $A_n@-$

モード：100

レジスタ： n

例 MOVE -(A3), \$3000

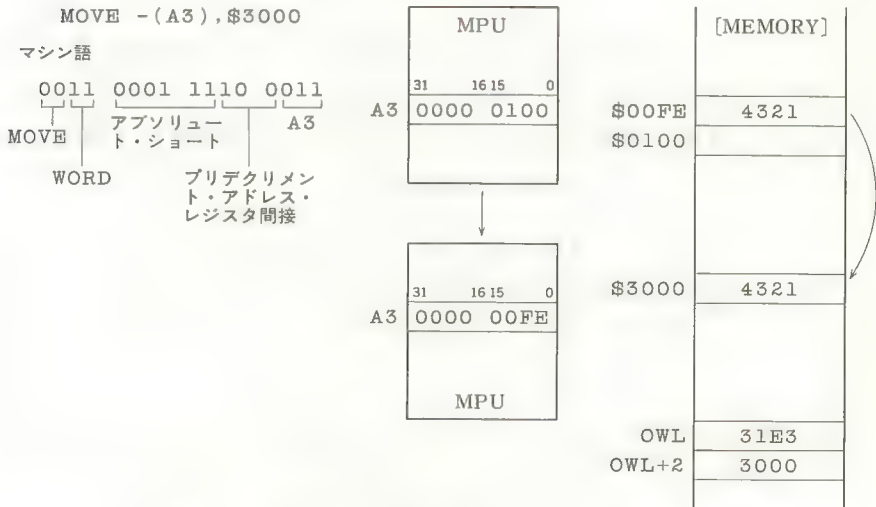


図 5・6 プリデクリメント付きアドレス・レジスタ間接

[4] オフセット付きアドレス・レジスタ間接 このアドレス・モードは、アドレス・レジスタの内容と16ビット変位付き整数の和がオペランド・アドレスを示します。

5 アドレッシング・モード

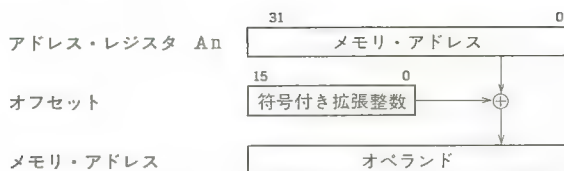


図 5・7

機能： $E A = (A_n) + d$

RTL 表記： $A_n @ (d)$

モード：101

レジスタ： n

アドレス・レジスタ A_0 に先頭アドレス“2000”とすると、命令を実行することにより、先頭アドレスにオフセット“200”が加わり、実効アドレス“\$2200”となって、アドレス内容“1234”をアドレス“\$4000”番地に転送します。

〔5〕インデックスおよびオフセット付きアドレス・レジスタ間接 このアドレス・モードは、アドレス・レジスタと符号拡張された8ビットの変位とインデックス・レジスタの和がオペランド・レジスタを示します。図5・8のようなフォーマットの1ワードの拡張ワードが必要です。



図 5・8

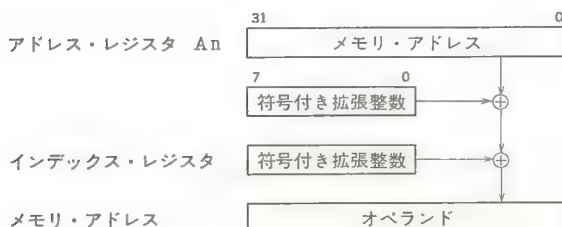


図 5・9

機能: $EA = (An) + (Ri) + d$

RTL 表記: $An@(d, Ri, W)$

$An@(d, Ri, L)$

モード: 110

インデックスは 16 or 32 ビットが選択できます。

レジスタ: n

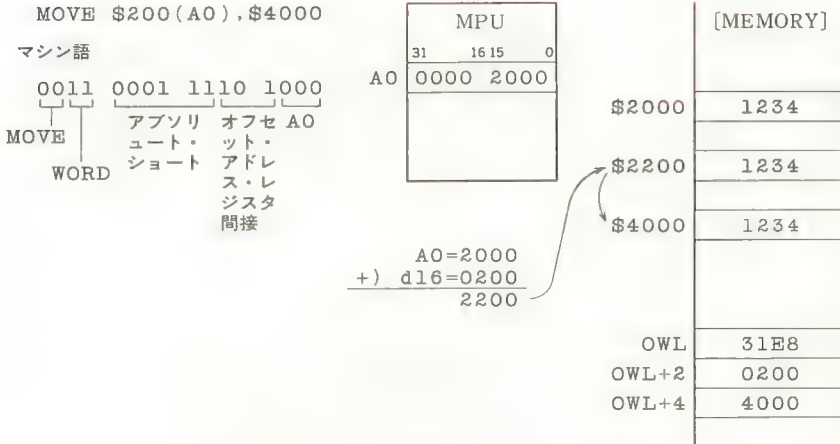


図 5・10 オフセット付きアドレス・レジスタ間接

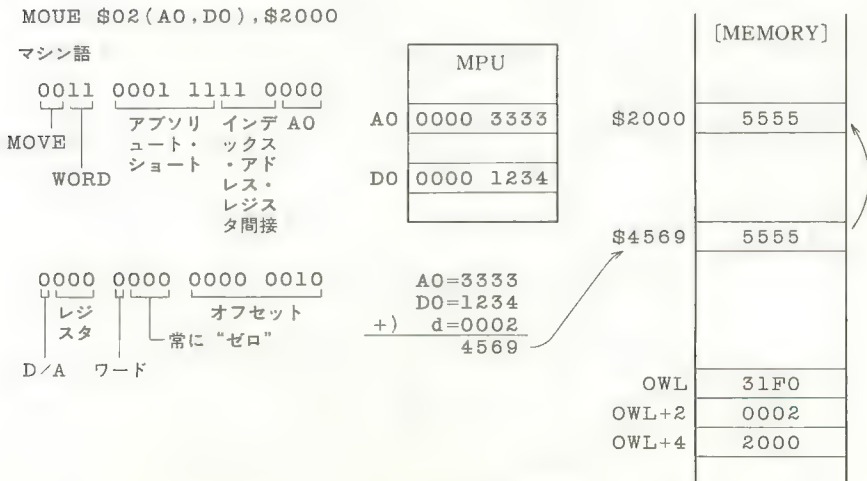


図 5・11 インデックスおよびオフセット付きアドレス・レジスタ間接

5.4 絶対(アブソリュート)モード

アブソリュート・アドレッシングは、ショートとロングの二つがあり、レジスタ番号の代わりに実効アドレス・フィールドを使用して、特定のアドレッシング・モードを指定します。

[1] アブソリュート・ショート このオペランド・アドレスは、拡張ワードに格納され、拡張ワードは16ビットであり、命令実行前に符号により拡張されます。このアドレッシング・モードは、1ワードの拡張ワードが必要です。

機能: EA given

RTL 表記: XXX.W

モード: 111

レジスタ: 000

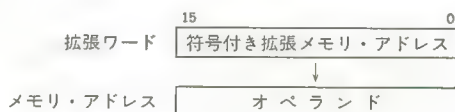


図 5.12 アブソリュート・ショート

[2] アブソリュート・ロング このアドレッシング・モードは、操作ワードの後の2ワードの拡張ワードによりオペランド・アドレスを示します。

機能: EA given

RTL 表記: XXX.L

モード: 111

レジスタ: 001

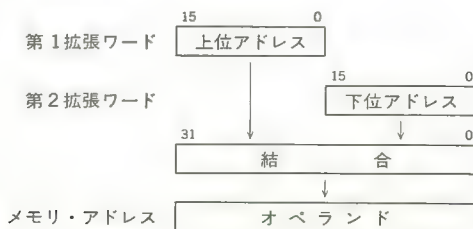


図 5.13 アブソリュート・ロング

5 アドレッシング・モード

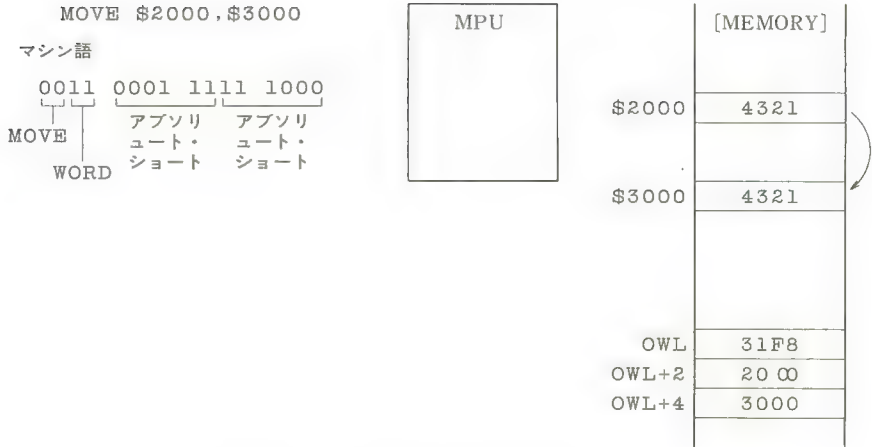


図 5・14 絶対（アブソリュート）ショート・アドレス

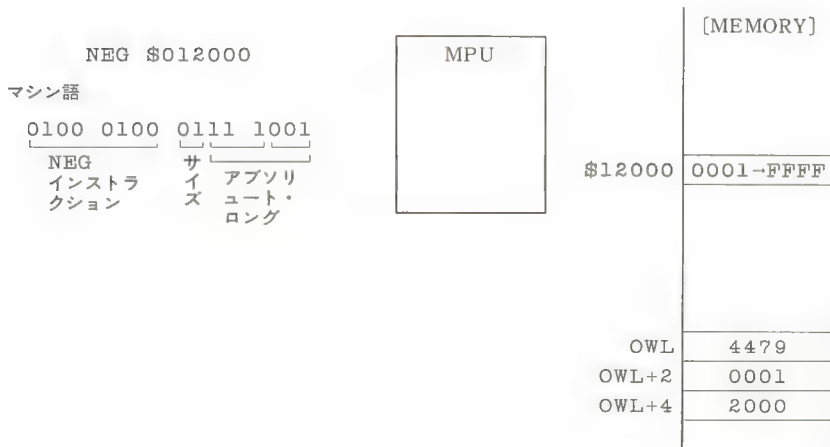


図 5・15 絶対（アブソリュート）ロング・アドレス

5.5 プログラム・カウンタ相対モード

プログラム・カウンタ相対モードは、プログラム・カウンタからの変位がオフセットにより決まるものと、オフセットとインデックスの両方によって決まるものの2種類のモードがあり、いずれもプログラミングにおける再配置手法やメモリ内のプログラム実行をフレキシビリティに使用できます。

〔1〕 オフセット付きモード 実行アドレスは、プログラム・カウンタと符号拡張された16ビット変位の和で示されます。また、このアドレッシング・モードは1ワードの拡張ワードが必要です。

機能： $EA = (PC) + d$

RTL 表記： $PC @ (d)$

モード：111

レジスタ：010

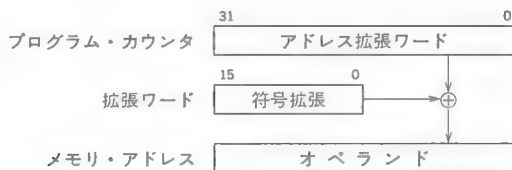


図 5・16

〔2〕 インデックスとオフセット付きモード 実行アドレスは、プログラム・カウンタと拡張された8ビット変位整数とインデックス・レジスタの和で示されます。また、プログラム・カウンタの値は拡張ワードのアドレスを示します。このアドレッシング・モードは1ワードの拡張ワードが必要です。

〔例〕 `MOVE<LABEL>, DO`

この命令を実行することにより、アセンブラが自動的にアドレスを計算し、



図 5・17

5 アドレッシング・モード

68000 は LABEL の付いた \$9002 番地の内容 “4321” をデータ・レジスタ DO の下位 16 ビットにローディングします。

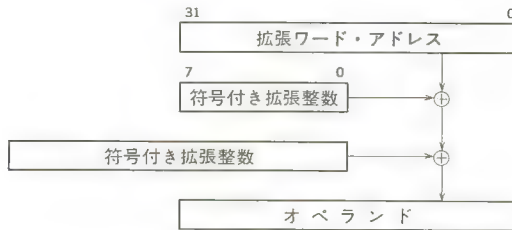


図 5・18

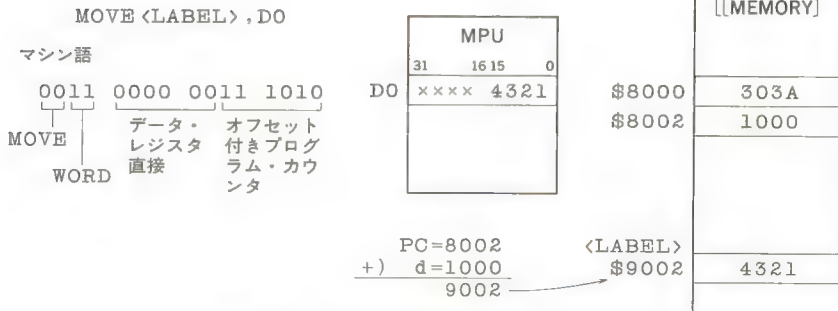


図 5・19 オフセット付きプログラム・カウンタ

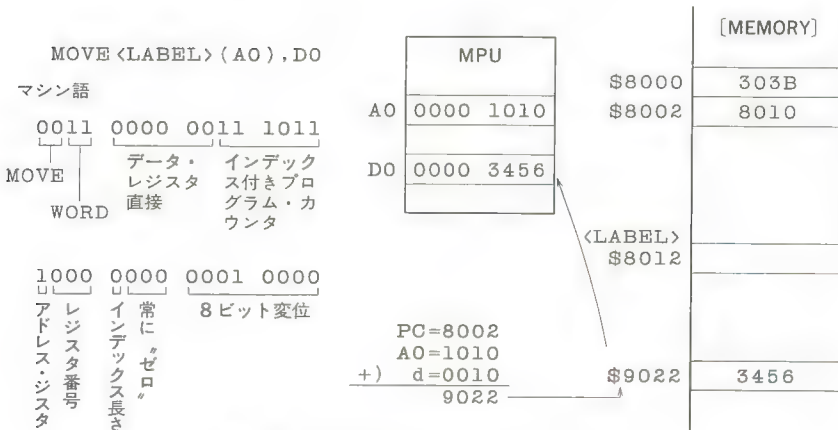


図 5・20 インデックス付きプログラム・カウンタ

5.6 イミディエート・モード

イミディエート・モードは、バイト・オペレーション、ワード・オペレーション、ロング・ワード・オペレーションの3タイプがあり、操作されるデータのサイズに応じて、1ワードまたは2ワードの拡張ワードを用います。また、アセンブラは、RTL 表記の“#”の文字を識別することによりイミディエート・データと判断します。

機能：オペランドで与えられる。

RTL 表記：#xxxx

モード：111

レジスタ：000

[例] `MOVE # $1000, A0`

ワード・データ \$1000 をアドレス・レジスタ A0 に \$00001000 がローディングされる。

クイック・イミディエート：クイック・イミディエート・モードの使用可能な命令は、ADDQ (add quick), MOVEQ (move quick), SUBQ (subtract quick) の3種類の命令です。

[例] `MOVEQ # $3A, D3`

バイト・データの \$3A がデータ・レジスタ D3 の下位 8 ビットに D3 (データ・レジスタの内容 “\$0000003A”) にローディングされます。

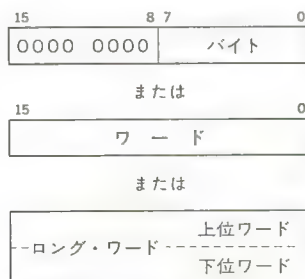


図 5・21

5 アドレッシング・モード

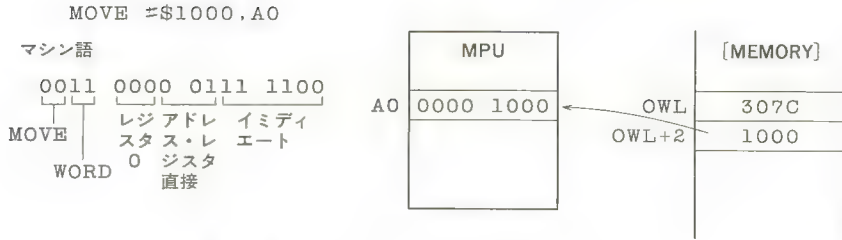


図 5.22 イミディエート・アドレッシング・モード

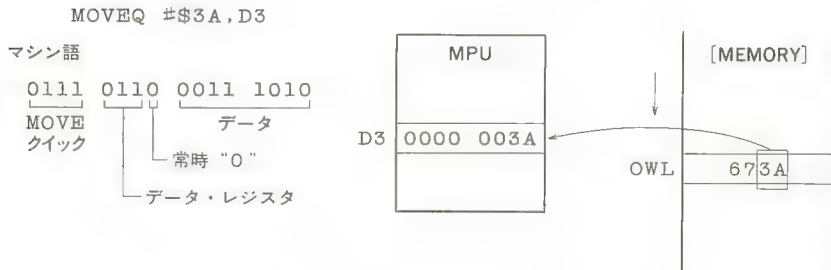


図 5.23 クイック・イミディエート・モード

5 アドレッシング・モード

5.7 インプライド・モード

インプライド・アドレッシング・モードは、暗黙的に指定できる命令があります。つまり、プログラム・カウンタ (PC)、システム・スタック・ポインタ (SP)、スーパーバイザ・スタック・ポインタ (SSP)、ユーザ・スタック・ポインタ (USP)、ステータス・レジスタ (SR) などです。

ブランチ命令のアドレッシングは、リラティブ・アドレッシングでオペレーション・ワード内に8ビットのディスプレースメント (2の補数表示) があり、これが“0”の場合は1ワードの拡張を必要とし、16ビット変位 (2の補数表示) となります。



図 5.24 分岐命令フォーマット

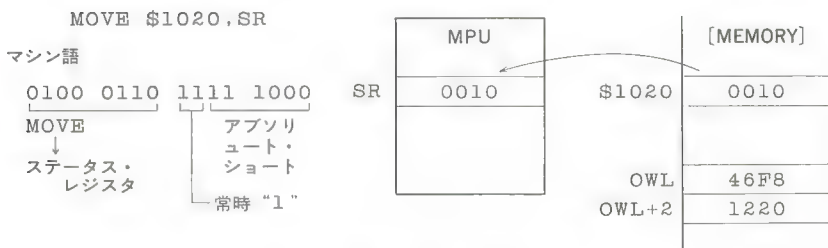


図 5.25 ステータス・レジスタ直接

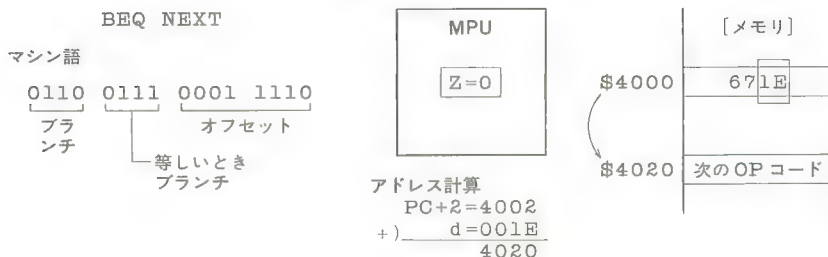


図 5.26 8ビット変位ブランチ (フォワード時)

5 アドレッシング・モード

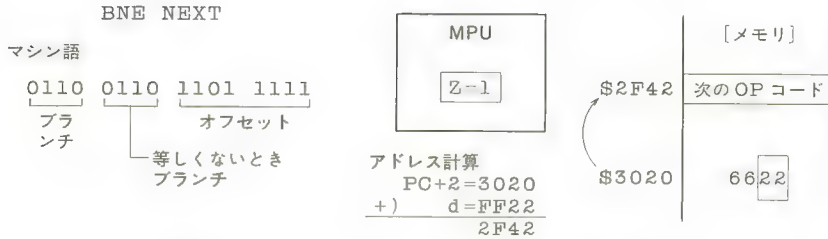


図 5・27 8ビット変位ブランチ（バックワード時）

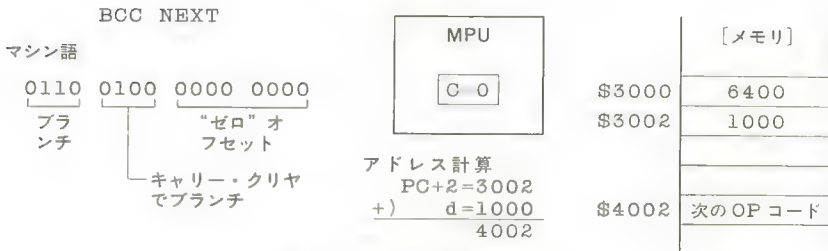


図 5・28 16ビット変位のとき

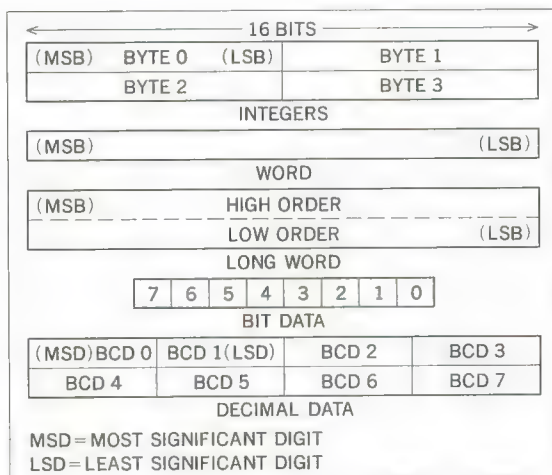
インプリシット命令一覧表

インストラクション	インプライド・レジスタ
Branch Conditional (Bcc), Branch Always (BRA)	PC
Branch to Subroutine (BSR)	PC, SP
Check Register against Bounds (CHK)	SSP, SR
Test Condition, Decrement and Branch (DBcc)	PC
Signed Divide (DIVS)	SSP, SR
Unsigned Divide (DIVU)	SSP, SR
Jump (JMP)	PC
Jump to Subroutine (JSR)	PC, SP
Link and Allocate (LINK)	SP
Move Condition Codes (MOVE CCR)	SR
Move Status Register (MOVE SR)	SR
Move User Stack Pointer (MOVE USP)	USP
Push Effective Address (PEA)	SP
Return from Exception (RTE)	PC, SP, SR
Return and Restore Condition Codes (RTR)	PC, SP, SR
Return from Subroutine (RTS)	PC, SP
Trap (TRAP)	SSP, SR
Trap on Overflow (TRAPV)	SSP, SR
Unlink (UNLK)	SP

5 アドレッシング・モード

基本的なデータ形式

1. 1 ビット (ビット)
2. 4 ビット (BCD デジット)
3. 8 ビット (バイト)
4. 16 ビット (ワード)
5. 32 ビット (ロング・ワード)



データ長とモードの関係

モード \ データ・サイズ	バイト	ワード	ロング・ワード
ポストインクリメント	+1	+2	+4
プリデクリメント	1	2	4

6. 例外処理

システムの正常動作のチェックと異常動作時の処理は、システムの信頼性を左右する重要な要素です。68000 は、豊富な例外処理機能を備え、この点を十分にカバーすることを説明しています。例外処理の種類と内容や割込みの機能、そしてベクタやアドレスの関係を説明します。

6.1 例 外 処 理

68000 の処理状態は、ノーマル処理、停止（ホルト）、例外処理の3方法がとられており、そのうちのどれかの処理状態にあります。68000 の異常動作の処理方法や割込み処理方法として例外処理（エクセプション・プロセッシング）があり、この方法によって68000 の高効率化と高信頼性を得ています。

例外処理には**内部起因による例外**と**外部起因による例外**の2種類があり、内部要因例外とは、アドレス・エラー、不正命令、特権違反などがあり、外部要因例外は、割込み、バス・エラー、リセットなどです。また、例外処理は3グループに分類され、256種類の定義（例外ベクタ割当て）があり、それぞれ固有の優先度とベクタ番号が割り当てられています（表6.1、表6.2 参照）。

プロセッサは、例外処理が生ずると例外処理ルーチンのアドレスをフェッチ（メモリ・ロケーション）してジャンプする（例外ベクタ）。図6.1に示すように、例外ベクタは2ワード長（リセット・ベクタを除く）であり、ベクタ番号は8ビットで表し、このベクタ番号を4倍にして例外ベクタのアドレスを作ります。図6.3に示すようにプロセッサは、8ビットのベクタ番号を24ビットのアドレスに変換します。

〔備考〕 割込みは、周辺デバイスから出されるプロセッサへの動作要求であり、

表 6.1 例外処理の分類と優先度

グループ	例外種類	処 理
0	リセット バス・エラー アドレス・エラー	実行中の命令はアボートされ、すぐに例外処理を開始する
1	トレース 割込み 不正命令 特権違反	実行中の命令が完結した後、次の命令をフェッチして例外処理を開始する
2	TRAP 命令 TRAPV 命令 CHK 命令 0 による割算	ノーマルの命令処理として例外処理に入る

表 6・2 例外ベクタの割当て

ベクタ番号	ア ド レ ス			割 当 て
	10 進	16 進	参照区分 ¹⁾	
0	0	000	SP	リセット：SSP の初期値
	4	004	SP	リセット：PC の初期値
2	8	008	SD	バス・エラー
3	12	00C	SD	アドレス・エラー
4	16	010	SD	不正命令
5	20	014	SD	ゼロによる割算
6	24	018	SD	CHK 命令
7	28	01C	SD	TRAPV 命令
8	32	020	SD	特権違反
9	36	024	SD	トレース
10	40	028	SD	1010 エミュレータ
11	44	02C	SD	1111 エミュレータ
12 ¹⁾	48	030	SD	(未定義)
13 ¹⁾	52	034	SD	(未定義)
14 ¹⁾	56	038	SD	(未定義)
15 ¹⁾	60	03C	SD	(未定義)
16～23 ¹⁾	64	040	SD	(未定義)
	95	05F		—
24	96	060	SD	スプリアス割込み
25	100	064	SD	自動ベクタ割込み (レベル 1)
26	104	068	SD	自動ベクタ割込み (レベル 2)
27	108	06C	SD	自動ベクタ割込み (レベル 3)
28	112	070	SD	自動ベクタ割込み (レベル 4)
29	116	074	SD	自動ベクタ割込み (レベル 5)
30	120	078	SD	自動ベクタ割込み (レベル 6)
31	124	07C	SD	自動ベクタ割込み (レベル 7)
32～47	128	080	SD	TRAP 命令
	191	0BF		—
48～63 ¹⁾	192	0C0	SD	(未定義)
	255	0FF		—
64～255	256	100	SD	ユーザ割込み
	1023	3FF		—

注 1) ベクタ番号 12～23, 48～63 は、現在使用されていないが、将来使用される可能性があるため、ユーザの周辺デバイスにこれらの番号 (アドレス) を割当てできない。

2) 参照区分の SP はスーパーバイザ・プログラム, SD はスーパーバイザ・データ。

6 例 外 処 理

バス・エラーとリセット入力は、アクセス・コントロールとプロセッサのリスタートに使用されます。

ノーマル処理は、命令実行に関するものです。メモリ・レファレンスによって命令およびオペランドをフェッチし、その結果をストアする処理です。

停止（ホルト）処理は、致命的なハードウェア上の異常により発生します。例

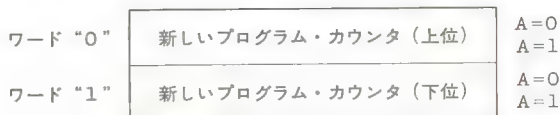


図 6・1 例外ベクタのフォーマット

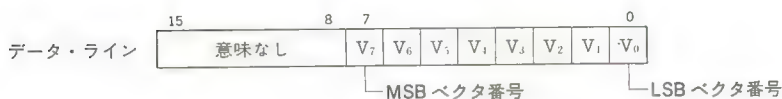


図 6・2 ベクタ番号のフォーマット



図 6・3 8ビットのベクタ番号から交換されるアドレス

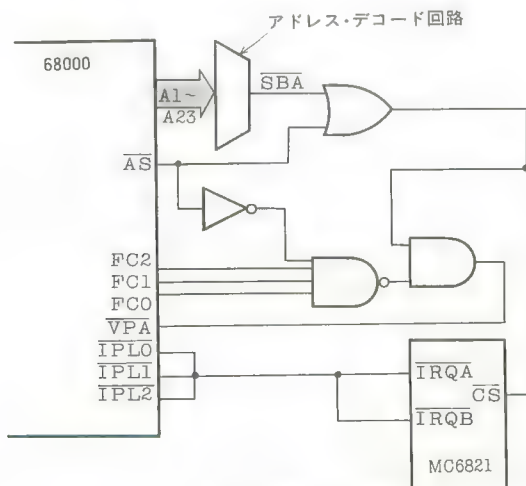


図 6・4 簡単な割込みストラクチャ

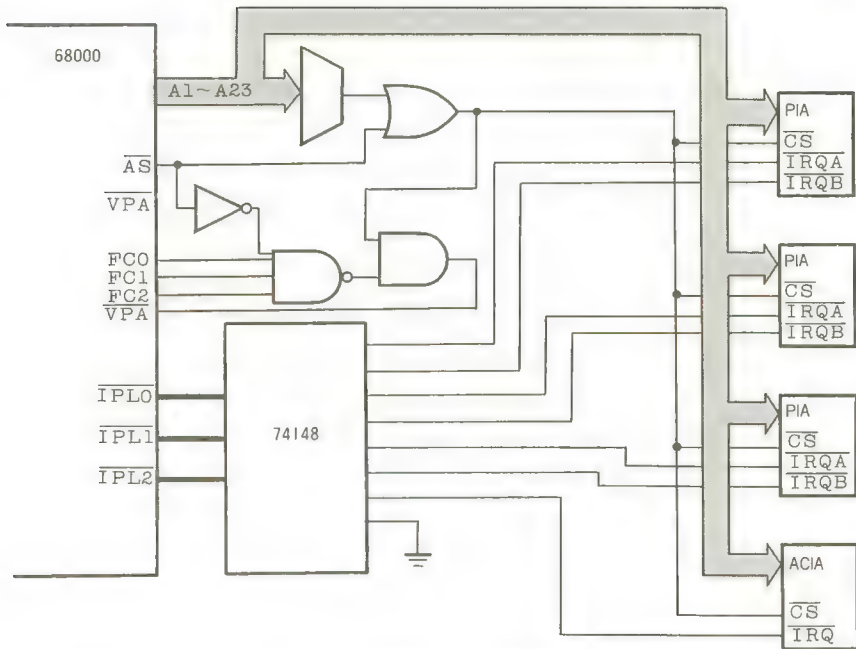


図 6・5 6800 周辺マルチ割込み

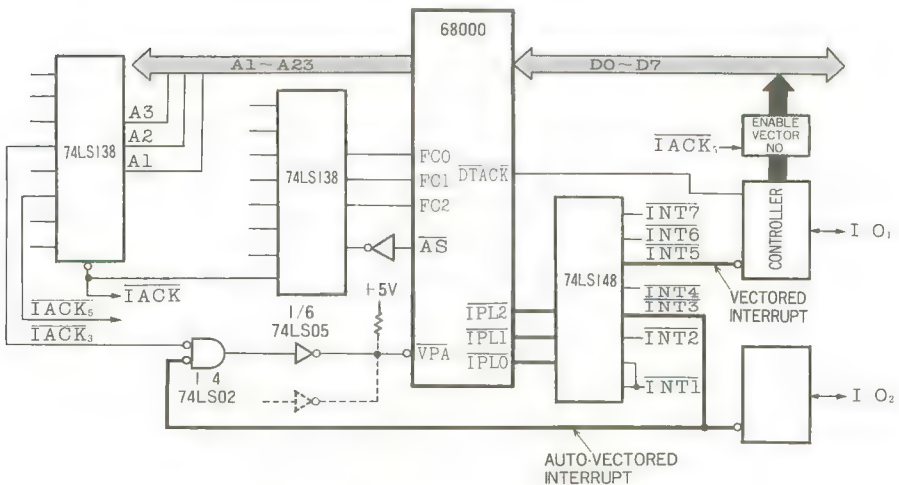


図 6・6 オート・ベクタ割込みとベクタ割込み回路

6 例 外 処 理

として、バス・エラーの例外処理中にさらに別のバス・エラーが発生したときなどです。

ベクタ番号のゼネレーション 図 6・7 に示す回路は、オートベクタで必ず用いられるベクタ番号発生回路のブロック図です。192 のユーザ割込みベクタは、64 から 255 までの一連のベクタ番号を参照して決められます。そして、これらのベクタ番号は、割込み優先度に従っておのこの割込みが振り分けられます。ベクタ番号 64 は最も低い優先度であり、ベクタ番号 255 は最も高い優先度です。この 192 レベルの外部で発生する優先度は 8 ビット (00000000 ~ 10111111) によって表され、外部発生優先度に 64 のオフセットをかけ、優先割込みエンコーディングによってゼネレートされます。

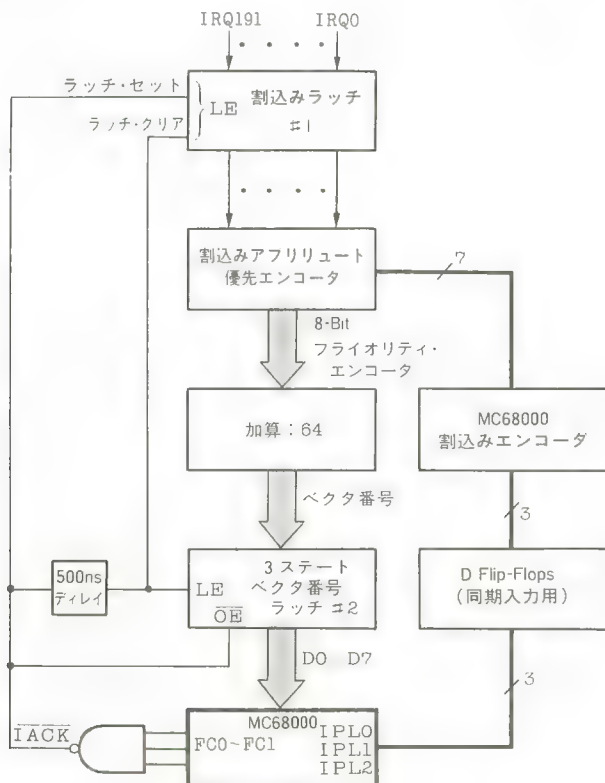
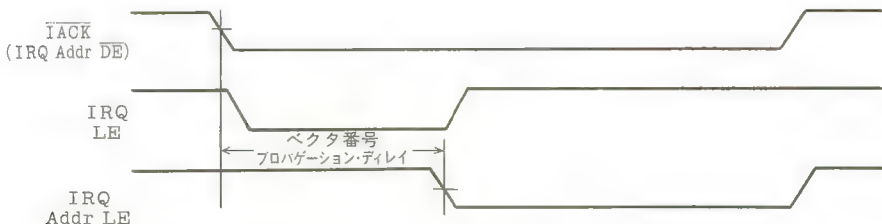


図6・9は、ベクタ番号発生回路のスキマティック・ダイアグラムを示します。この図は、192種類の割込みベクタ番号を供給するためのものであり、192本の割込み要求線は、オクタル・ラッチ用のSN74LS373と3～8のエンコーダSN74LS148を通して8グループに構成され、それぞれ24区分されているスキマティックです。そして、8レベルの優先度とおおのこのグループへの割込みは、3ビットのエンコーダ信号線のA0, A1, A2によって割り振られます。

図6・10に示す割込みエンコーディング回路は、割込み処理を行う方法の回路例です。プライオリティ・エンコーダからのすべてのバリッド割込み入力は、プロセッサのIPLピンで表示されます。そして、プライオリティ・エンコーダが動作状態のときには、プロセッサがIPLピンをサンプルして、現行のプロセッサ・レベルより高いかどうかを判断します。

例えば、システムがレベル2で動作しているときには、レベル3割込みをプライオリティ・エンコーダは保持し、現行命令実行サイクル終了後にレベル3割込みが実行されます。また、レベル3割込みが実行される前にレベル4の割込みが受け付けられた場合は、プライオリティ・エンコーダの出力を短時間でグリッチし、レベル4の出力へと変更させます。

割込み入力は、SN74LS273の8ビット・レジスタにてラッチされ、SN74LS348の8～3ラインのプライオリティ・エンコーダでもってコンバートされます。フリップ・フロップSN74LS175は、SN74LS273とSN74LS348を通すことによるディレイ・タイムとIPL入力ピンへのインプット・セットアップのためのものです。



DTACK デレイ:ベクタ番号プロパゲーション・ディレイ-235ns

図 6・8 ベクタ番号発生回路のタイミング図

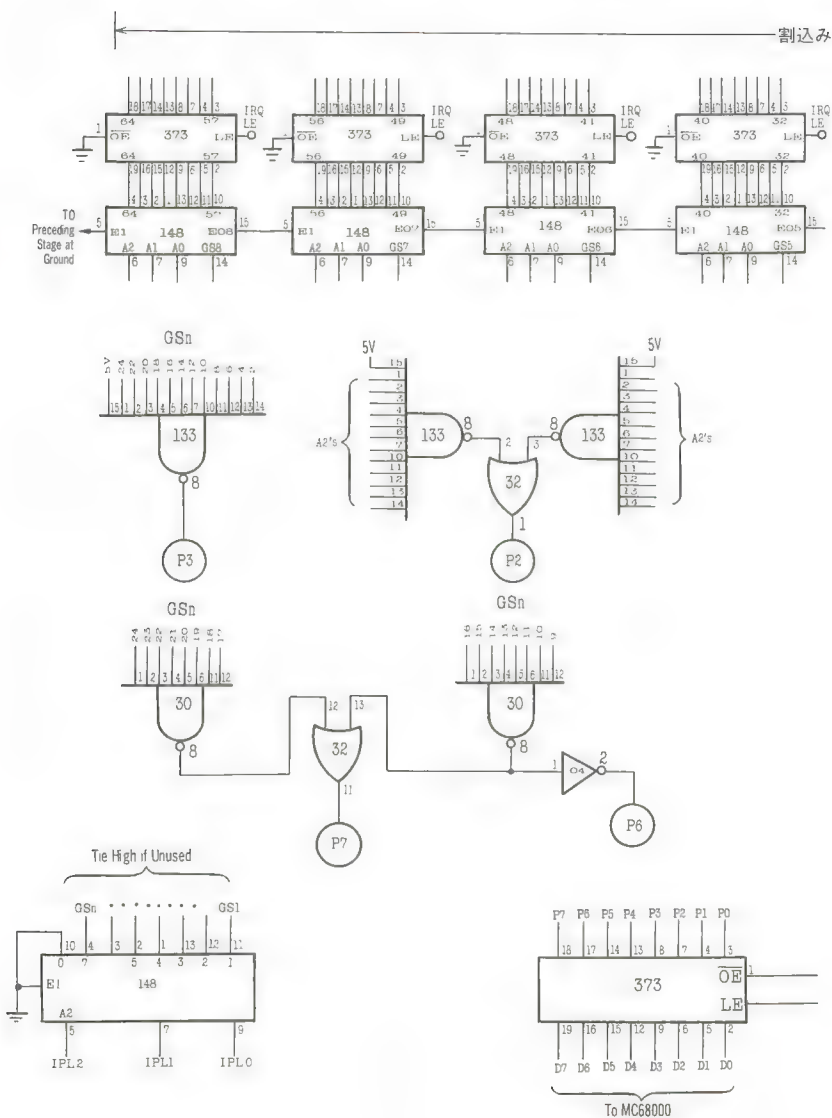
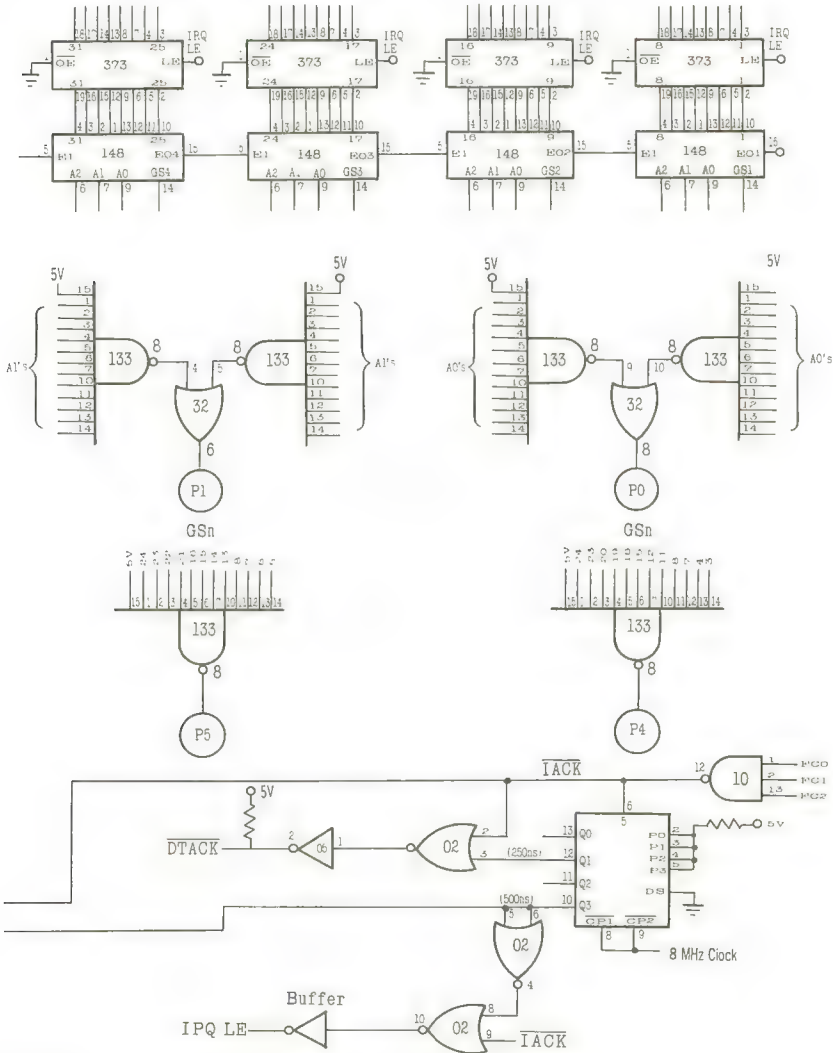


図 6・9 ペクタ

入力 1~64



番号発生回路例

6 例 外 処 理

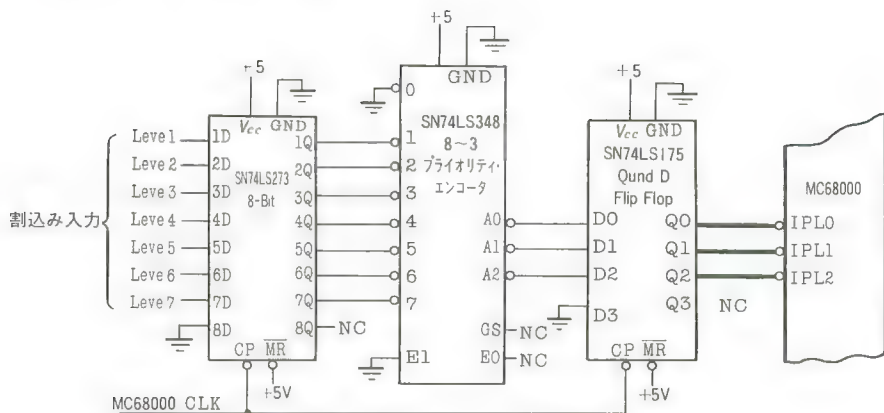


図 6・10 割込みエンコーディング回路例

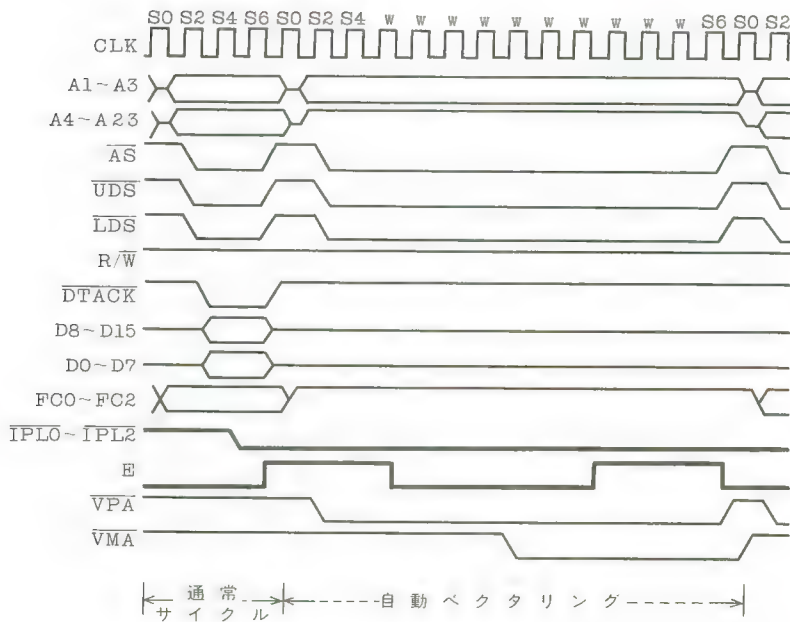


図 6・11 自動ベクタ・タイミング

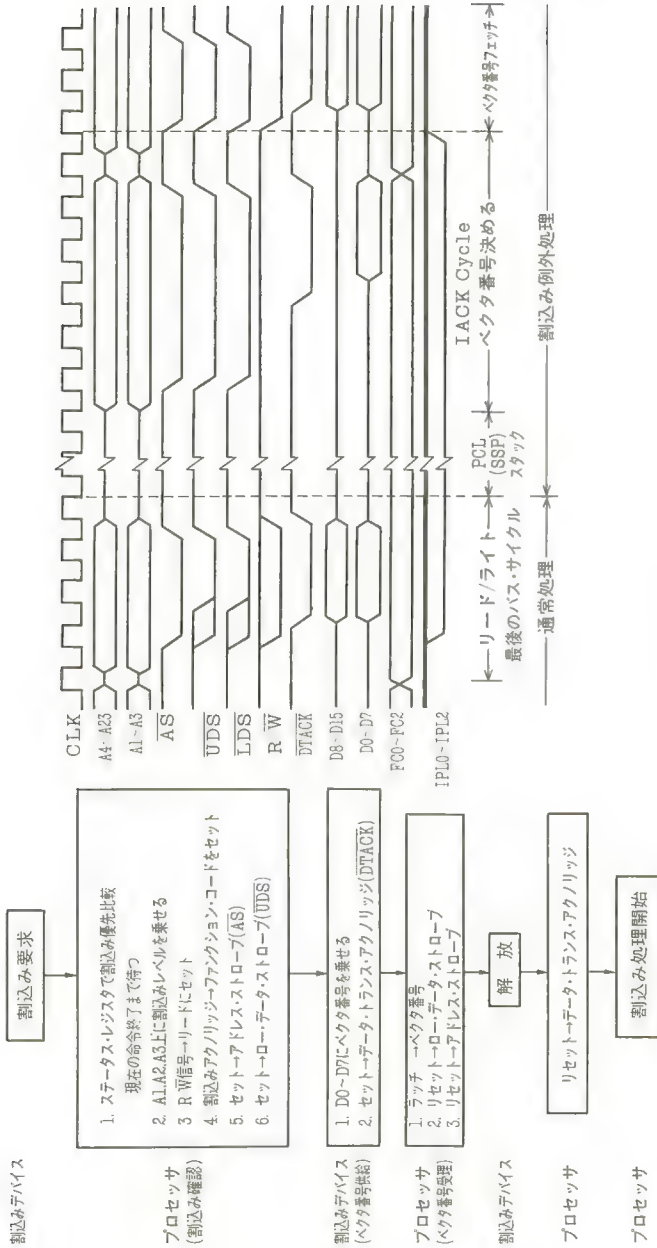


図 6-12 割込み処理タイミング

6・2 例外処理シーケンス

例外処理（エクセプション・プロセッシング）シーケンスは、そのエクセプションの起こる原因により、第1ステップ、第2ステップ、第3ステップ、第4ステップと四つのステップを踏みます。例外処理時における例外処理シーケンスフローを図6・13に示します。

〔1〕 **第1ステップ** 68000 内部のステータス・レジスタへ一時的に待避し、Sビット（スーパーバイザ・モード）がセットし、プロセッサはスーパーバイザ特権状態になります。また、Tビット（トレース・モード）はリセットされて、例外処理ルーチンに入ります。なお、リセットや割り込み例外では、割り込み優先度マスクも更新され、現在実行しようとする割り込み処理が終了するまでは、処理中の割り込み優先度以下の割り込みは受け付けられません。

〔2〕 **第2ステップ** 例外処理ベクタ番号が決定されます。また、割り込みでのベクタ番号をプロセッサはフェッチします。ベクタ・アドレスは、ベクタ番号に4が乗じられて変換され、256個のベクタ番号を識別します。

〔3〕 **第3ステップ** 現在のプロセッサの内容（SRの値やPCの値）がスーパーバイザ・スタック・ポインタを用いて収納されます。収納されたプログラム・カウンタの値は、まだ実行されていない次の命令を指します。

〔4〕 **第4ステップ** プロセッサの内容が一新されて、プロセッサはベクタ・テーブルから先頭番地を求めて命令例外処理の実行を開始します。通常はRET命令（RET from Exception）で終了させます。

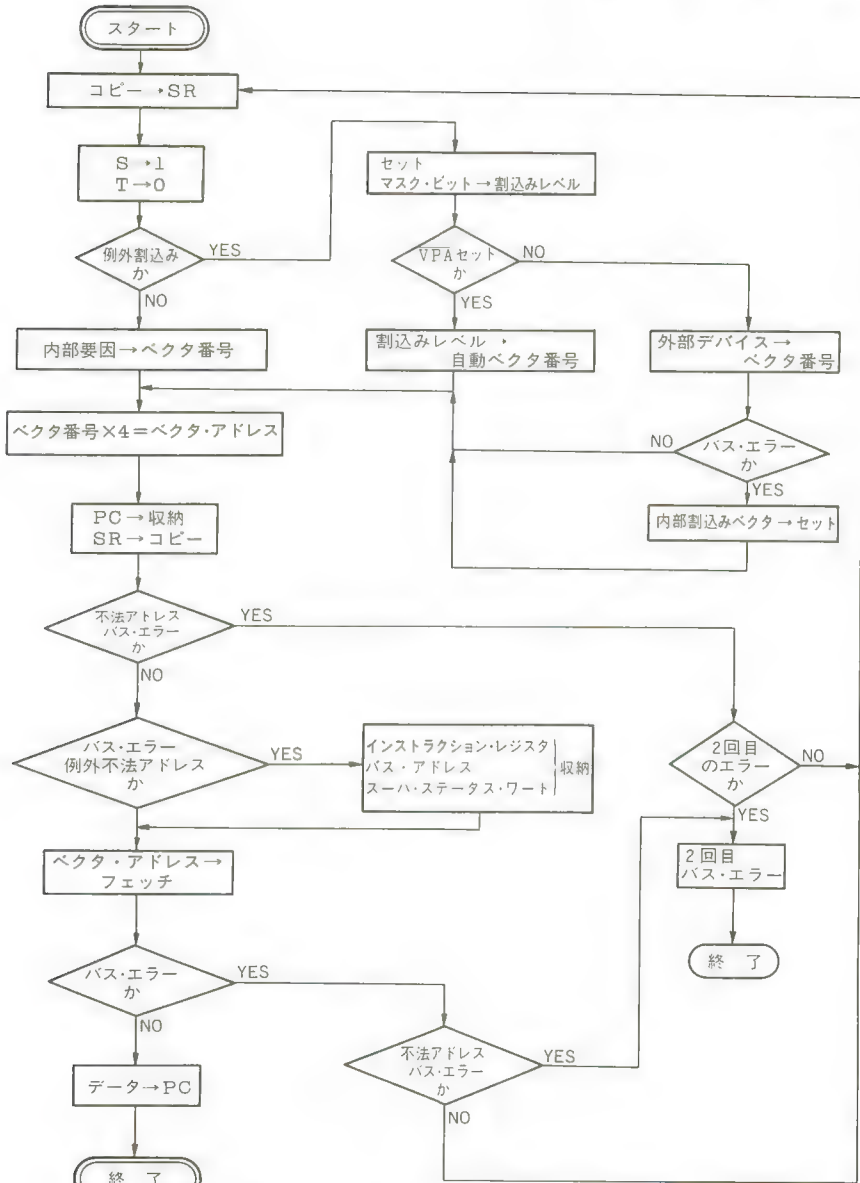


図 6・13 例外プロセッシング・シーケンス

6.3 バス・エラー

バス・エラーは、外部デバイスとハンドシェイクを行うことを前提として、外部回路はアドレス・ストローブ (\overline{AS}) とデータ転送アクノリッジ (\overline{DTAK}) 間の時間を測定し、エラーか正常かを判断し、バス・エラー (\overline{BERR}) 信号を発生させます。プロセッサは、バス・エラー信号を受信すると、バス・エラー例外シーケンスを開始するか、バス・サイクルを再実行 (リラン) するかを決めます。また、バス・エラーやアドレス・エラーやリセットの例外処理中にバス・エラーが発生した場合は、プロセッサはホルトになり、メモリ内容を保護するためにすべての処理を中止します。なお、ホルトしたプロセッサの解除は、外部からのリセット入力信号によって行います。表 6.3 にバス・エラー信号と機能の関係を示しました。

[1] バス・エラー例外シーケンス プロセッサがバス・エラー信号 (\overline{BERR}) を受信し、ホルト (\overline{HALT}) がセットされていないときにバス・エラー例外シーケンスに入ります。

- 処理手順
- 1) プログラム・カウンタとステータス・レジスタを収納。
 - 2) エラー情報を収納。
 - 3) バス・エラーのベクタ開始テーブルを読む。
 - 4) バス・エラーのルーチンを実行する。

(バス・エラーのベクタ番号は 2 でアドレスは 008)

[2] バス・サイクルのリラン プロセッサがバス・エラー信号 (\overline{BERR}) を受信し、ホルト (\overline{HALT}) が外部デバイスによってセットされているとき、プ

表 6.3 バス・エラー信号と機能の関係

\overline{BERR}	\overline{HALT}	機 能
0	0	現在のサイクルを再実行
0	1	例外処理
1	0	シングル・バス・サイクル・オペレーション
1	1	ノーマル・オペレーション

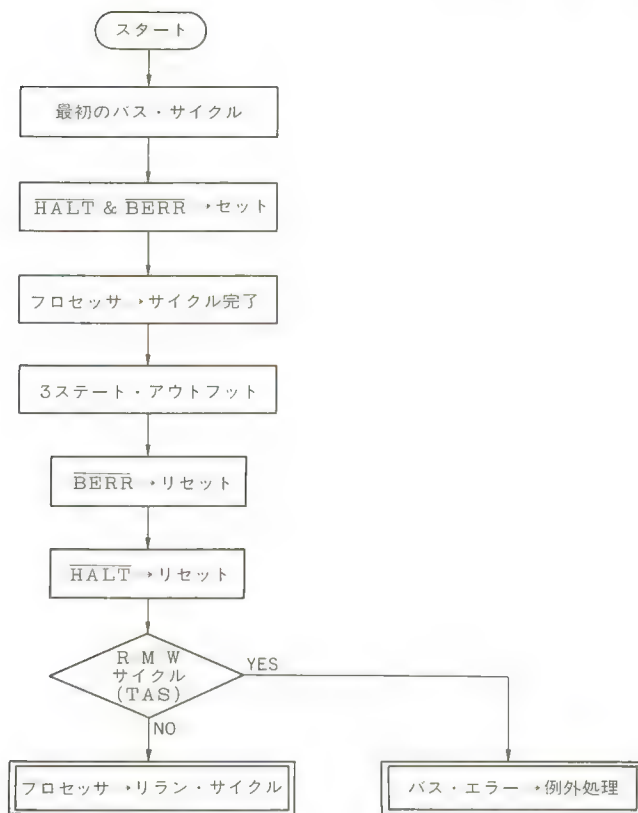


図 6・14 リラン・サイクル・フロー

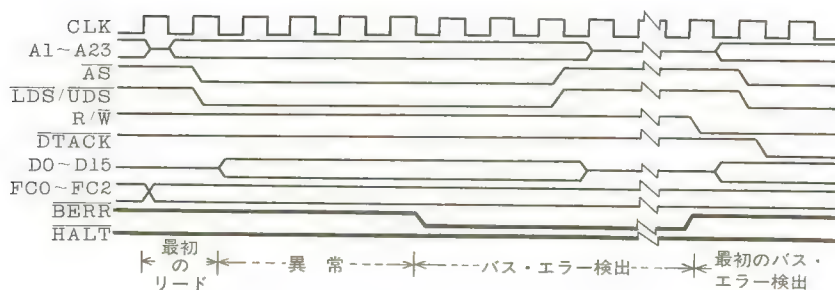


図 6・15 バス・エラーのタイミング

6 例 外 処 理

ロセッサはリラン・シーケンスに入ります。

プロセッサがホルト状態で外部回路によってホルト信号がリセットされるまでは、次のバス・サイクルは実行されません。ホルト信号がリセットされると、プロセッサは、エラーを起こしたときと同じアドレス、同じファンクション・コード、同じデータ、同じコントロール信号を使用して、前と同じバス・サイクルを再実行（リラン）します。

アドレス・エラー例外は、ワード・アクセスやロング・ワード・アクセス時に起きたり、奇数アドレスのインストラクション時に起こります。アドレス・エラーが生ずると、バス・サイクルはアボードされ、内部バス・エラーを発生させます。そして、プロセッサは、現在の処理を終了させてから例外処理を開始します。例

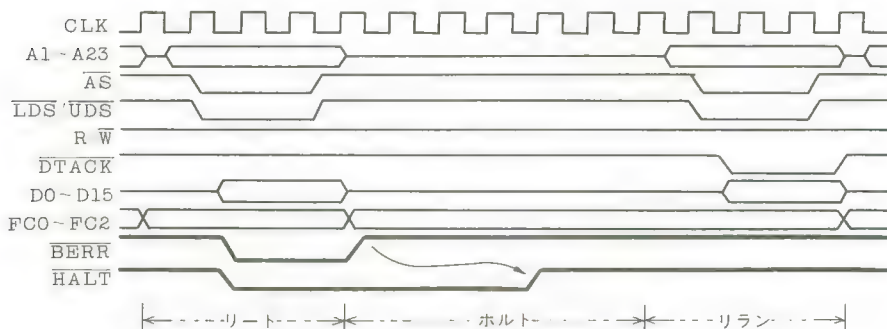


図 6-16 リラン・バス・サイクルのタイミング

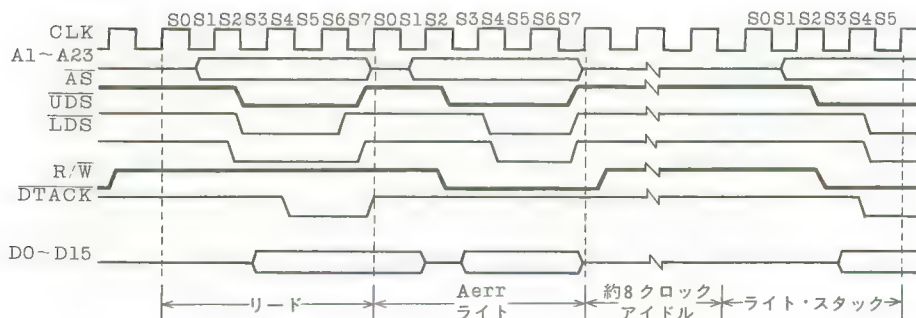


図 6-17 アドレス・エラーのタイミング

6 例 外 処 理

外処理開始後のシーケンスは、バス・エラーと同様に実行されます。ただし、アドレス・エラー・ベクタを参照し、ベクタ番号を格納することを除きます。また、アドレス・エラーがバス・エラー例外処理中に起こったときには、プロセッサはホルトになります。

図 6・17 は、アドレス・エラー時の例外処理によってフォローされるショート・バス・サイクルの実行タイミング図です。

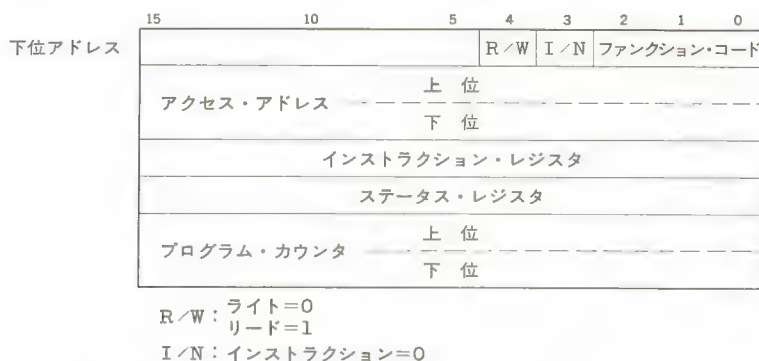


図 6・18 バス・アドレス・エラー・エクセプションのためのスーパーバイザ・スタック配列

6.4 リセットとトラップ

外部要因によってリセットが発生すると、進行中の処理はすべてクリアされ、S ビットがセットし、T ビットはリセットします。そして割込み優先度は最も高い“7”であり、プロセッサは強制的にスーパーバイザ状態にセットされます。図 6・19 に示すように、例外ベクタ領域；ベクタ番号 0 におけるベクタ・アドレスは、最初の 4 バイト（2 ワード）の値がスーパーバイザ・スタック・ポインタの初期値（\$00200000）となり、残りの 4 バイト（2 ワード）の値がプログラム・カウンタの初期値（\$00E00100）となります。

その後は、プログラム・カウンタで示されるアドレスから命令の実行を開始します。また、電源投入やリスタート・シーケンスにおけるプログラム・カウンタの初期の先頭アドレスを指定する必要があります。

ソフトウェアによる RESET 命令では、リセット・ベクタをローディングせずにリセット・ラインをセットして、外部デバイスをリセットします。リセット後は、RESET 命令の次の命令から実行されてゆきます。

トラップは、命令に起因する例外です。プロセッサが命令実行中に異常な状態を認めたか、ノーマル動作としてトラップを発生させるような命令を使用して発生するものの 2 とおりがあります。ステータス・レジスタとプログラム・カウンタはスーパーバイザ・スタックに退避され、例外ベクタに格納されているベクタ・アドレスにより命令の実行を開始します。

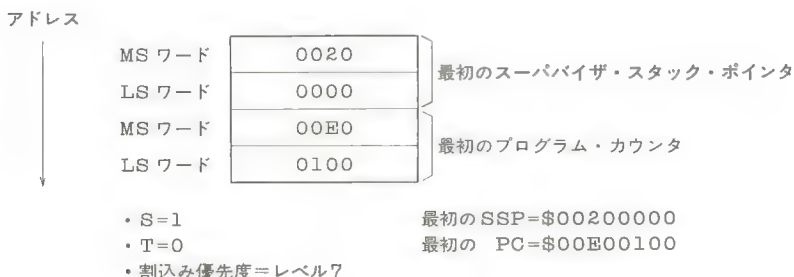


図 6・19 リセット例外ベクタ

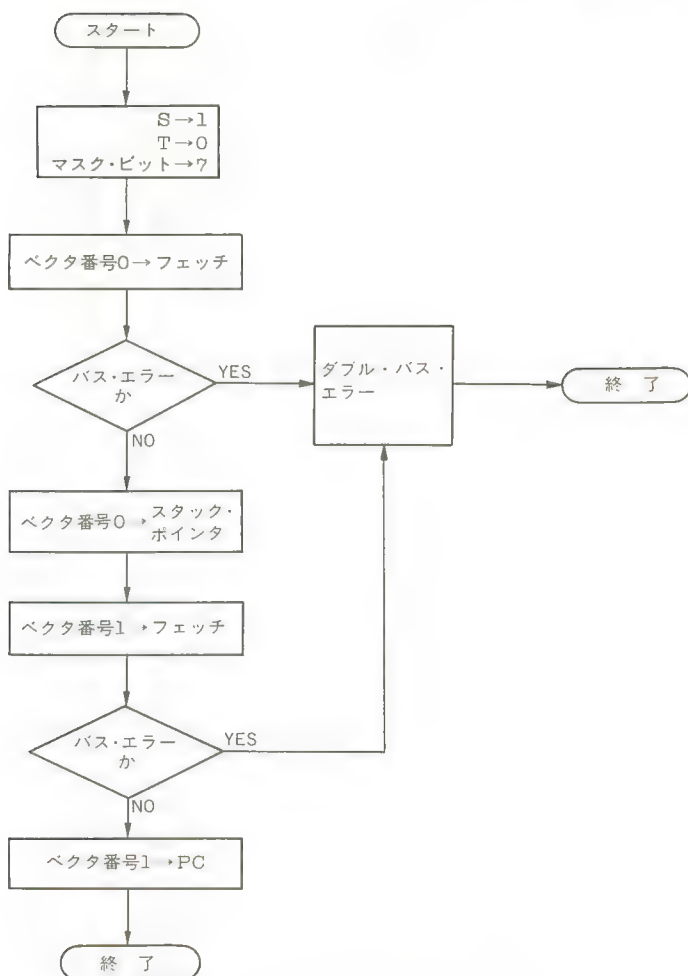


図 6・20 リセット例外プロセッシング・フロー

表 6・4 トラップに関する命令と機能

命 令	機 能
TRAPV	エラー・チェック用
CHK	エラーが発生したときだけトラップが発生する
DIVS	エラー・チェック用
DIVU	エラーが発生したときだけトラップが発生する
TRAPO	常に例外処理を発生させる
TRAP15	システム・コール、ブレーク・ポイント、エミュレートなどに用いる

6.5 特 権 状 態

特権状態には、ユーザ状態とスーパーバイザ状態の2種類があり、命令レファレンスには、スーパーバイザ・スタック (SSP) とユーザ・スタック・ポインタの2とおりがあります。

特権状態は、マイクロプロセッサ・システムの安全性を確保するためのものであり、特権機構におけるユーザ状態での大部分のプログラム実行の安全性を持たせ、外部メモリ管理ユニット (MMU) によるアクセス制御やアドレス変換などに使用します。

ユーザ状態でのアクセスは制御されるので、システムの他の領域へのアクセスは許されません。スーパーバイザ状態では、オペレーティング・システム (OS) の管轄のもとに 68000 の持つ全命令が実行可能であり、ユーザ状態のプログラム管理タスクの実行ができます。

スーパーバイザ状態は、特権状態の中でも上位に位置し、ステータス・レジスタの S ビットがセットされるとプロセッサはスーパーバイザ状態になります。このときのバス・サイクルは、スーパーバイザ・レファレンスとしてスタック操作をスー

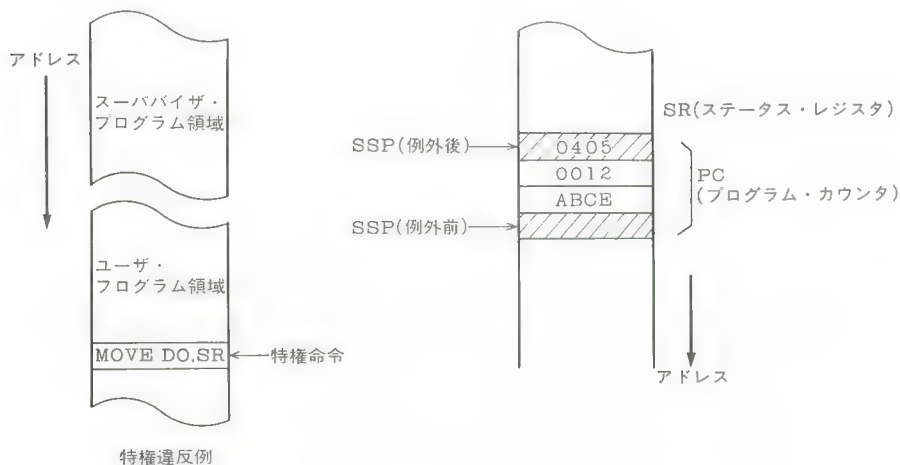


図 6・21 例外処理のスタック方法

パイザ・スタック・ポインタ (SSP) にします。

ユーザ状態は特権状態の中でも下位に位置し、ステータス・レジスタの **S** ビットがリセットされていると、プロセッサはユーザ状態で命令を実行します。バス・サイクルは、ユーザ状態レファレンスとして、スタック操作とシステム・スタックポインタ、またはユーザ・スタック・ポインタ (USP) にします。なお、システムに対して重要な影響を与える命令は、ユーザ状態では実行できません。また、デバッグ・プログラムに用いるユーザ・スタック・ポインタへの移動命令 (**MOVE to USP**) や、ユーザ・スタック・ポインタからの移動命令 (**MOVE from USP**) もユーザ状態では実行できません。

ベクタ・アドレス		ベクタ番号	
HEX		HEX	
64	0010 2346	19	
68	0020 0000	1A	
6C	VPA #3	1B	
70	VPA #4	1C	
74	VPA #5	1D	
78	VPA #6	1E	
7C	VPA #7	1F	
\$102346	VPA #1 SERVICE ROUTINE		
\$200000	VPA #2 SERVICE ROUTINE		

図 6・22 例外ベクタの例

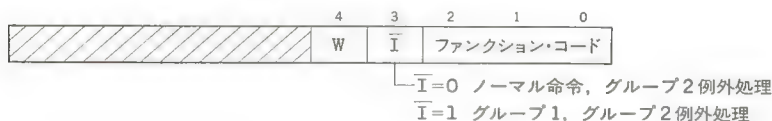
6.6 特権違反とトレース

特権命令は、システムを安全に動作させるための命令であり、次の八つの命令があります。STOP, RESET, RTE, MOVE USP, MOVE to SR, AND Immediate to SR, EOR Immediate to SR, OR Immediate to SR です。

特権違反時の例外処理は、命令をデコードした後に、プロセッサが特権違反かどうかを判断して、例外処理の開始を行います。また、命令のオペレーション・ワードのビット・パターンが正規なものと異なったビット・パターンの命令を**違法命令**といいます。

トレース・モードは、ステータス・レジスタ (SR) の T ビットをセットすることによって、現在の命令実行終了後にトレース例外の割込みを起し、ユーザ・モードのプログラムを 1 命令ごとに実行をモニタしたり解析したりすることができます。そして、命令実行後モニタへ制御を戻すことができます。T ビットがリセットされているときは、トレース・モードに入らず、ノーマルな命令を次々と実行します。

割込み優先度による命令の不履行や違法命令の命令、特権命令などは、トレース・モードには入りません。また、例外時や、アドレス・バス・エラーなどによって命令が中断されたときも、トレース・モードはできません。



スーパー・ステータス・ワード
上位バス・アドレス
下位バス・アドレス
インストラクション・レジスタ
ステータス・ワード
上位プログラム・カウンタ
下位プログラム・カウンタ

図 6.23 違法アドレスとバス・エラーのスタッキング

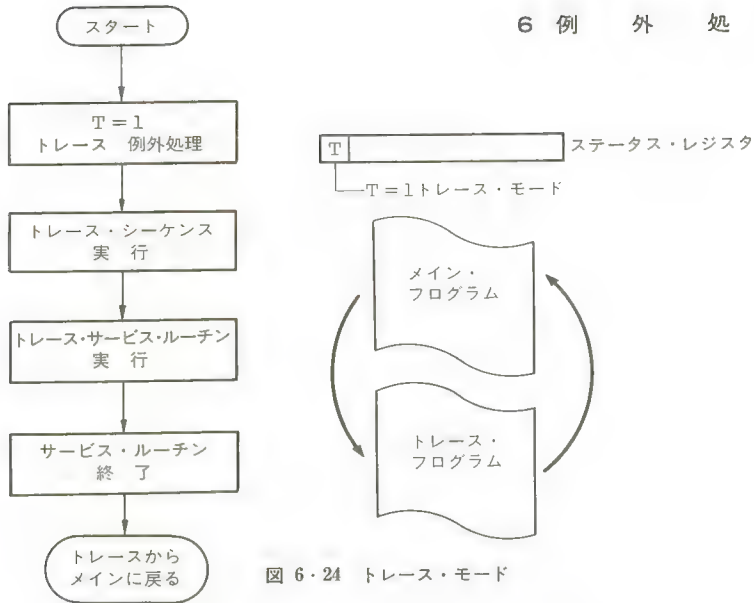


図 6・24 トレース・モード

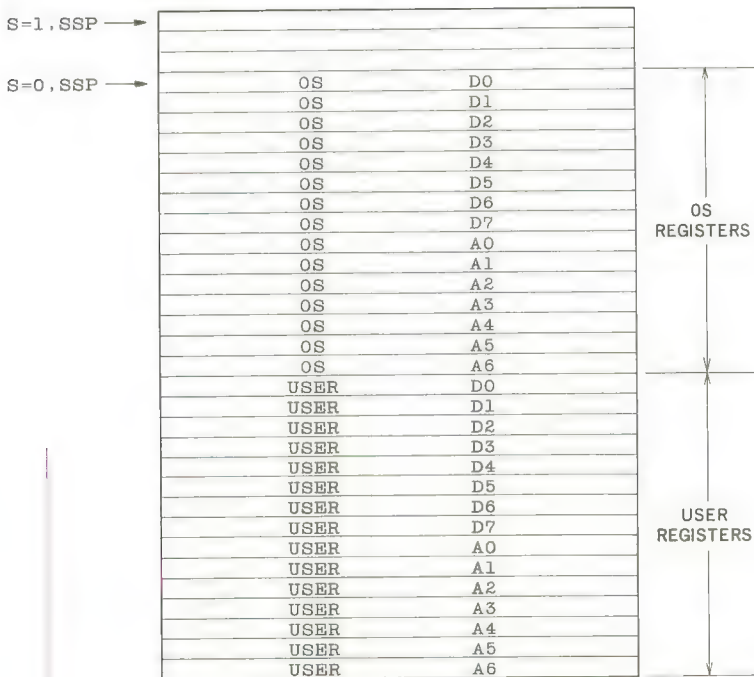


図 6・25 トレース・ストレージ

6 例 外 処 理

TRACE:1 MC68000 ASM

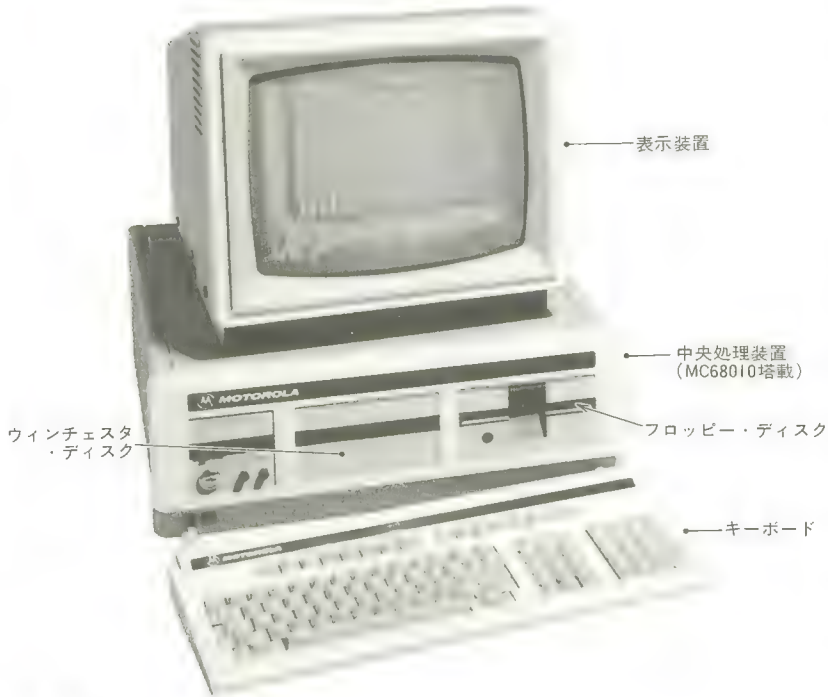
```

10                                *   TRACING THE USER PROGRAM
20                                *
30                                *THIS ROUTINE TRACES A SPECIFIED NUMBER
40                                *OF INSTRUCTIONS IN A USER PROGRAM.
50                                *INSTRUCTIONS EXECUTED IN THE SUPER-
60                                *VISOR MODE ARE NOT TRACED.
70                                *
71                                00002000    RORG $2000
72                                *
73                                *ENTER OS;S=1
80                                *
80 002000 48EF7FFF0042    MOVEM.L D0-D7/A0-A6,$42(SP)
90 002008 4CEF7FFF0006    MOVEM.L 6(SP),D0-D7/A0-A6
91 00200C 43FA0024    LEA TRACEX,A1
92 002010 21C90024    MOVE.L A1,$24
100                                *
110                                *DO ANY REQUIRED OS TASKS. A TASK WHICH
120                                *MUST BE DONE IS TO INITIALIZE THE
130                                *TRACE TABLE WITH THE NUMBER OF IN-
140                                *STRUCTIONS TO BE TRACED, THE ADDRESS
150                                *FROM WHICH TRACING IS TO BEGIN, AND
160                                *THE DESIRED CONTENTS OF THE STATUS
170                                *REGISTER(T=1).
180                                *
190 002014 41F81000    LEA TABLE,A0
200 002018 3010    MOVE (A0),D0
210 00201A 2F6800020002    MOVE.L 2(A0),2(A7)
220 002020 3EAB0006    MOVE 6(A0),(A7)
230 002024 48EF7FFF0006    AGAIN MOVEM.L D0-D7/A0-A6,6(SP)
240 00202A 4CEF7FFF0042    MOVEM.L $42(SP),D0-D7/A0-A6
250 002030 4E73    RTE
260 002032 48EF7FFF0042    TRACEX MOVEM.L D0-D7/A0-A6,$42(SP)
270 002038 4CEF7FFF0006    MOVEM.L 6(SP),D0-D7/A0-A6
280                                *
290                                *TRACE SERVICE ROUTINE TASKS SHOULD
300                                *BE DONE HERE. DO IS BEING USED TO
310                                *COUNT THE NUMBER OF INSTRUCTIONS
320                                *WHICH HAVE BEEN TRACED.
330                                *
340 00203E 51C8FFE4    DBRA DO,AGAIN
350 002042 08970007    DONE BCLR 7,(SP)
360 002046 60FE    BRA *
370                                *TRACE TABLE INFORMATION
380                                ORG $1000
390 001000 0005    TABLE DC 5    * # OF INST. TO BE TRACED
400 001002 00005000    DC.L $5000    *ADDRESS OF USER PROGRAM
410 001006 8700    DC $8700    *INITIAL STATUS REGISTER
420                                *
430                                *TRACE VECTOR INITIALIZATION
440                                *
450                                00000024    ORG $24
460 000024 00002032    DC.L TRACEX
470                                *
480                                END
490
500
510
520

```

図 6・26 トレース・ユーザ・プログラム

VME/10システム基本構成



特 長

1. アプリケーション開発装置として使用できる。
2. OEM 用組み込み機器として使用できる。
3. VME モジュールと I/O モジュール用のカード・ケージを備えている。
4. 8/16/32 ビット・プロセッサを中心とする機器のハードウェアとソフトウェアの開発をサポート。
5. 8/16/32 ビットのエミュレータ・ステーションと連動することができる。
6. VERSAdos オペレーティングシステムによるサポート。
7. UNIX オペレーティング・システム (SYSTEM V/68) もサポート。
8. システム価格が低廉 (EXORmacs の 1/4)。

VME/10 マイクロコンピュータ・システム (その 1) <その 2 は p. 131>

68000 処理ステート	ノーマル	命令実行 (STOP 含む)
	例 外	インタラプト
		トラップ
		トレース
	ホ ル ト	ハードウェア・ホルト
		ダブル・バス・エラー
		ダブル・イリーガル
		アドレス・エラー
	外部要因	割込み (レベル 1~7)
		バス・エラー
リセット		
例 外 処 理	アドレス・エラー	
	トレース	
内部要因	TRAP, TRAPV, CHK, DIV 命令	
	違法命令	
	特権違反	
	バウンダリ・エラー	

7. 命令セット

MC68000 の命令は，設計の段階で使用頻度や命令の機械語へのコード化が配慮されています。そして，機能の高い命令セットや例外における回避や復帰の処理方法が強化され，第5章の豊富なアドレッシング・モードとの関連やレジスタとの組合せによって多様な使用方法が可能です。

MC68000 の命令を機能別に9個のグループに分け，その代表的な命令の機能を説明します。

7.1 命令形式と命令タイプ

〔1〕 命令フォーマット 68000 の命令は、1 ワードから 5 ワードの長さがあります。それぞれの命令の長さと実行される操作は、1 ワードのオペレーション・ワードとこれに続くイミディエート・オペランドの 2 ワード、または、オペレーション・ワードの実行アドレス・モードで指定されたオペランドの 2 ワードで、合計 5 ワードの命令が最長のものです。また、68000 のバイナリ・パターンは、このオペレーション・ワード内に含まれます (図 7.1、図 7.2 参照)。

オペレーション・ワードとオペランドは、偶数番地にアドレスしますので注意してください (奇数番地時にはアドレス・エラー・エクセプションが起きます)。

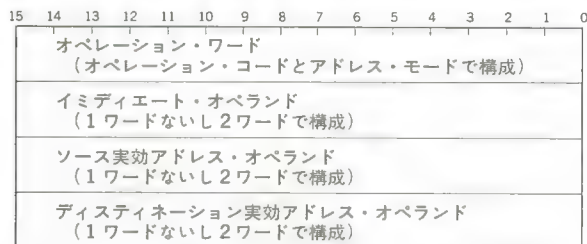


図 7.1 命令フォーマット

ニーモニック `MOVE.X <ea>, <ea>`

↑ ↑
ソース ディスティネーション

X はデータ長の指定で、次のとおり

B: バイト

W: ワード

L: ロング・ワード

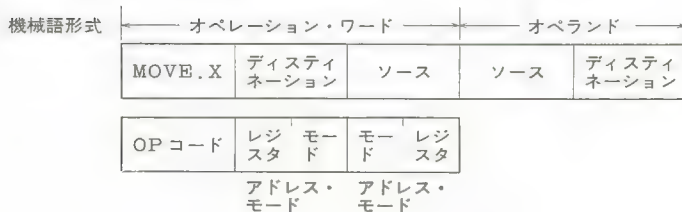


図 7.2 MOVE 命令の例

7 命 令 セ ッ ト

〔2〕 **命令タイプ** 68000 の基本命令タイプには56種類のタイプがあります。そのうちのバリエーションのタイプとして8種類があります（表7・1、表7・3参照）。

表 7・1 68000——56の基本命令セット——

ニーモニック	内 容	ニーモニック	内 容
ABCD	拡張付きアド・デシマル	NBCD	拡張付きネゲート・デシマル
ADD	アド	NEG	ネゲート
AND	ロジカル・アンド	NOP	ノー・オペレーション
ASL	算術シフト・レフト	NOT	1の補数
ASR	算術シフト・ライト	OR	ロジカルOR
BCC	ブランチ・コンディション	PEA	実効アドレス・プッシュ
BCHG	ビット・テストと変更	RESET	外部デバイス・リセット
BCLR	ビット・テストとクリア	ROL	拡張なしのローテート・レフト
BRA	常時ブランチ	ROR	拡張なしのローテート・ライト
BEST	ビット・テストとセット	ROXL	拡張付きローテート・レフト
BSR	サブルーチンへのブランチ	ROXR	拡張付きローテート・ライト
BTST	ビット・テスト	RTE	例外からのリターン
CHK	レジスタ限界チェック	RTR	リターンと再格納
CLR	オペランド・クリア	RTS	サブルーチンからのリターン
CMP	コンペア	SBCD	拡張付きデシマル割算
DBCC	コンディション・テスト、デクリメントとブランチ	SCC	コンディション・セット
DIVS	符号付き割算	STOP	ストップ
DIVU	符号なし割算	SUB	割算
EOR	エクスクルーシブOR	SWAP	レジスタ半分スワップ
EXG	レジスタ変更	TAS	テストとオペランド・セット
EXT	符号付き拡張	TRAP	トラップ
JMP	ジャンプ	TRAPV	オーバフローにおけるトラップ
SSR	サブルーチンへジャンプ	TST	テスト
LEA	実効アドレス・ロード	UNLK	アンリンク
LINK	スタック・リンク		
LSL	ロジカル・シフト・レフト		
LSR	ロジカル・シフト・ライト		
MOVE	ムーブ		
MOVEM	ムーブ・マルチ・レジスタ		
MOVEP	ムーブ・ペリフェラル・データ		
MULS	符号付きマルチプル		
MULU	符号なしマルチプル		

7 命 令 セ ッ ト

7.2 データ移動命令

データ移動命令は、表 7.2 に示すように汎用レジスタやメモリのデータなどを転送するのに用います。

MOVE (ムーブ・データ・フロム・ソース・トゥ・ディスティネーション) のオペレーション・サイズは、バイト、ワード、ロング・ワード長のデータを指定でき、データ・レジスタからデータ・レジスタへ、メモリとデータ・レジスタ間や、メモリからメモリへ転送します。また、データは転送時にチェックし、コンディションコードをセットします。

MOVEM (ムーブ・マルチプル・レジスタ) は、レジスタ・リストで指定した複数のレジスタにより、実効アドレスで指定したところから連続したメモリの位置へ転送したり、転送されたりします (3.3 節「スタックの構成」参照)。

MOVEP (ムーブ・ペリフェラル) は、データ・レジスタと 1 バイトおきのメモリの間でデータ (バイト) 転送を行います。データ・レジスタの偶数のアドレ

表 7.2 データ移動命令

命 令	オペランド・サイズ	機 能	内 容
EXG	32	$Rx \leftrightarrow Ry$	レジスタ交換
LEA	32	$EA \rightarrow An$	実行アドレス・ロード
LINK	—	$An \rightarrow SP@-$ $SP \rightarrow An$ $SP+d \rightarrow SP$	リンクと割付け
MOVE	8, 16, 32	$(EA)s \rightarrow EAd$	ソースから行先へデータ移動
MOVEM	16, 32	$(EA) \rightarrow An, Dn$ $An, Dn \rightarrow EA$	複数レジスタ移動
MOVEP	16, 32	$(EA) \rightarrow Dn$ $Dn \rightarrow EA$	周辺データ移動
MOVEQ	8	$\#imm \rightarrow Dn$	クイック移動
PEA	32	$EA \rightarrow SP@-$	実行アドレス格納
SWAP	32	$Dn[31:16] \leftrightarrow Dn[15:0]$	レジスタ半分スワップ
UNLK	—	$An \rightarrow SP$ $SP@+ \rightarrow An$	リンク解除

S: ソース, d: ディスティネーション, []: ビット番号

スのときはデータ・バスの上半分で転送され、奇数のアドレスのときはデータ・バスの下半分を用いて転送します。

MOVEQ (ムーブ・クイック) は、8ビットのイミディエート・データをディスティネーションのデータ・レジスタに移します。データは、32ビットのロング・ワードに符号拡張されて、データ・レジスタにロードされます。

EXG (エクスチェンジ・レジスタ) は、常にロング・ワード・オペレーションで二つのレジスタ内容を交換します。

表 7・3 8タイプの特定制のバリエーション型

命令タイプ	バリエーション	内 容
ADD	ADD ADDA ADDQ ADDI ADDX	加 算 アドレス加算 クイック加算 イミディエート加算 拡張付き加算
AND	AND ANDI	ロジカル・アンド イミディエート・アンド
CMP	CMP CMPA CMPM CMPI	比 較 アドレス比較 メモリ比較 イミディエート比較
EOR	EOR EORI	エクスクルーシブOR イミディエート・エクスクルーシブOR
MOVE	MOVE MOVEA MOVEQ MOVE from SR MOVE to SR MOVE to CCR MOVE to USP	転 送 アドレス転送 クイック転送 ステータス・レジスタからの転送 ステータス・レジスタへの転送 コンディション・コード・レジスタに転送 ユーザ・スタック・ポインタに転送
NEG	NEG NEGX	ネゲート 拡張付きネゲート
OR	OR ORI	ロジカルOR イミディエートOR
SUB	SUB SUBA SUBI SUBQ SUBX	割 算 アドレス割算 イミディエート割算 クイック割算 拡張付き割算

7 命 令 セ ッ ト

LEA (ロード・イフェクティブ・アドレス) は、ソース・オペランドの実効アドレスを指定したアドレス・レジスタにロードします。

PEA (プッシュ・イフェクティブ・アドレス) は、スーパバイザ・スタックかユーザ・スタックかのいずれかに実行アドレスが計算されて、スタックにプッシュされます。

SWAP (スワップ・レジスタ・ハーフ) は、データ・レジスタの上位 16 ビットと下位 16 ビットの内容を入れ替えることができます。

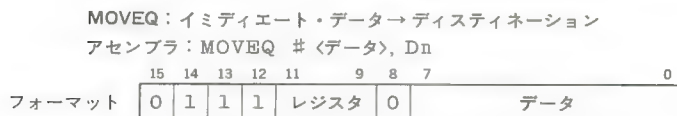
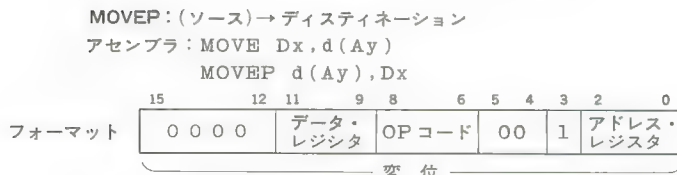
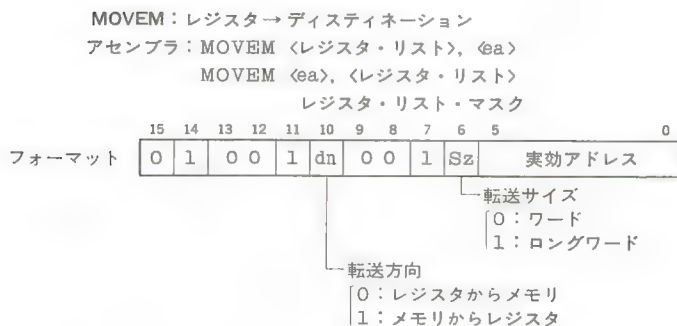
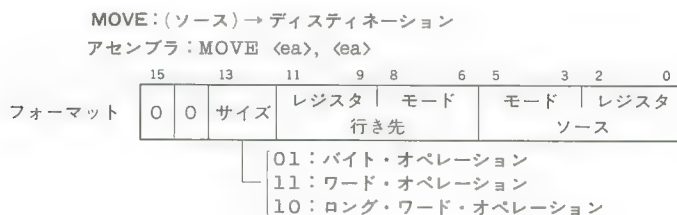


図 7・3 (1) データ移動命令フォーマット

7 命 令 セ ッ ト

EXG: Rx ↔ Ry

アセンブラ: EXG Rx, Ry

	15	13	11	8	7	3	2	0
フォーマット	11	00	Rx レジスタ	1	OP モード		Ry レジスタ	

LEA: ディスティネーション → An

アセンブラ: LEA <ea>, An

	15	14	13	12	11	9	8	7	6	5	0
フォーマット	0	1	00	レジスタ		111			実効アドレス		

PEA: ディスティネーション → SP@-

アセンブラ: PEA <ea>

	15	14	13	11	10	6	5	0
フォーマット	0	1	00	1	0000	1	実効アドレス	

SWAP: レジスタ [31:16] ↔ レジスタ [15:0]

アセンブラ: SWAP Dn

	15	14	13	11	10	6	5	2	0
フォーマット	0	1	00	1	0000	1	000	レジスタ	

図 7・3 (2) データ移動フォーマット

MOVE DO, D1	データ・レジスタ D0 の下位ワードを D1 の下位ワードに転送
MOVE.L \$01000, \$02000	\$1000 番地のロングワードを \$2000 番地に転送
MOVE.W #'A', 3000	値 'A'00 を \$3000 番地へ転送
MOVE \$1000, A1	\$1000 番地のワードをアドレス・レジスタ A1 の下位ワードへ転送

図 7・4 MOVE 命令の例

7 命 令 セ ッ ト

```
BLMOVE:1  MC68000 ASM

1          *   BLOCK MOVE
2          *
3          *MOVE A BLOCK OF DATA FROM
4          *SOURCE TO DESTINATION. D0 CONTAINS
5          *THE NUMBER OF LONG WORDS-1
6          *TO BE MOVED. A0 CONTAINS THE SOURCE
7          *ADDRESS AND A1 CONTAINS THE
8          *DESTINATION ADDRESS.
9          *
10         ORG $1000
11 001000 22D8      AGAIN MOVE.L (A0)+,(A1)+
12 001002 51C8FFFC  DBRA D0,AGAIN
13          END

***** TOTAL ERRORS  0--  0

SYMBOL TABLE

AGAIN      001000
```

図 7・5 ブロック転送のプログラム例

7.3 整数算術演算命令

表7.4に示すように、加算、減算、乗算、除算、比較などを二つのオペランドを用いて行うことができます。そして、クリアやテストなどの単一オペランドでも演算が行えます。

表 7.4 整数算術演算命令

命 令	オペランド・サイズ	機 能	内 容
ADD	8, 16, 32 16, 32	$Dn + (EA) \rightarrow Dn$ $(EA) + Dn \rightarrow EA$ $(EA) + \#xxx \rightarrow EA$ $An + (EA) \rightarrow An$	2進加算
ADDX	8, 16, 32 16, 32	$Dx + D6 + X \rightarrow Dx$ $Ax @ - Ay @ + X \rightarrow Ax @$	拡張加算
CLR	8, 16, 32	$0 \rightarrow EA$	クリア
CMP	8, 16, 32 16, 32	$Dn - (EA)$ $(EA) - \#xxx$ $Ax @ + - Ay @ +$ $An - (EA)$	比 較
DIVS	32÷16	$Dn \div (EA) \rightarrow Dn$	符号付き除算
DIVU	32÷16	$Dn \div (EA) \rightarrow Dn$	符号なし除算
EXT	8→16 16→32	$(Dn) 8 \rightarrow Dn 16$ $(Dn) 16 \rightarrow Dn 32$	符号拡張
MULS	16*16→32	$Dn * (EA) \rightarrow Dn$	符号付き乗算
MULU	16*16→32	$Dn * (EA) \rightarrow Dn$	符号なし乗算
NEG	8, 16, 32	$0 - (EA) \rightarrow EA$	補 数
NEGX	8, 16, 32	$0 - (EA) - X \rightarrow EA$	拡張付き補数
SUB	8, 16, 32 16, 32	$Dn - (EA) \rightarrow Dn$ $(EA) - Dn \rightarrow EA$ $(EA) - \#xxx \rightarrow EA$ $An - (EA) \rightarrow An$	2進減算
SUBX	8, 16, 32	$Dx - Dy - X \rightarrow Dx$ $Ax @ - - Ay @ - - X \rightarrow Ax @$	拡張付き減算
TAS	8	$(EA) - 0, 1 \rightarrow EA[7]$	オペランド・テストとセット
TST	8, 16, 32	$(EA) - 0$	オペランド・テスト

[] ビット番号

7 命 令 セ ッ ト

ADD (アト・バイナリ) は、ソース・オペランドをディスティネーション・オペランドに加算し、結果をディスティネーションに収納します。オペレーション・サイズはバイト、ワード、ロング・ワードがあり、各データ・オペランドに対して演算を行います。加算命令には、**ADDA** (アドレス加算)、**ADDI** (イミディエート加算)、**ADDQ** (クイック加算) などのバリエーションがあります。

ADDX (アト・エクステンド) は、二つの方法によるアドレッシングによってソース・オペランドと拡張ビット (**X**) が行先オペランドに加算されてディスティネーションに収納されます。多倍長数、つまり 32 ビット以上の長さを持つデータの加算に便利です。

減算は、**SUB** (減算)、**SUBA** (アドレス減算)、**SUBI** (イミディエート減算)、**SUBQ** (クイック減算) などのバリエーションを持ち、オペレーション・サイズは、バイト、ワード、ロング・ワードを持ちます。

SUB (サブトラクト・バイナリ) は、ソース・オペランドをディスティネーション・オペランドから減算し、結果をディスティネーションに収納します。

SUBX (サブトラクト・ウィズ・エクステンド) は、二つの方法によるアドレッシングによって、ソース・オペランドと拡張ビット (**X**) をディスティネーション・オペランドから減算し、結果をディスティネーションに収納します。多倍長数、つまり 32 ビット以上の減算に便利です。

CMP (コンペア) は、ソース・オペランドとデータ・レジスタの内容を比較して、コンディション・コードをセットします。オペレーション・サイズは、バイト、ワード、ロング・ワードがあり、**CMPA** (アドレス比較)、**CMPI** (イミディエート比較)、**CMPI** (メモリ比較) があります。

MULS (サイン・マルチプイ)、**MULL** (アンサイン・マルチプイ) は、2 個の符号付きおよび 2 個の符号なしの 16 ビット整数のオペランドを乗算し、32 ビットの積をデータ・レジスタに入れます。また、レジスタのオペランドは下位ワードしか使用しません。

DIVS (サイン・デバイド)、**DIVU** (アンサイン・デバイド) は、ディスティネーションのオペランドであるロング・ワード (32 ビット) の被除数を、ソース・オペランドであるワード (16 ビット) の除数で除算し、その結果を 16 ビット商

は上位ワードへ、16ビット余りを下位ワードで示し、ディスティネーションに収納されます。

EXT (サイン・エクステンド) は、データ・レジスタの最上位ビットである符号ビットを、バイトからワードへ、またはワードからロング・ワードへ拡張します。

TAS (テスト・アンド・セット) は、指定されたバイト・オペランドをテストし、その結果をコンディション・コードの **N** (負フラグ) と **Z** (ゼロ・フラグ) にセットし、オペランドの最上位ビットを“1”にします。また、有効な利用方法としてマルチプロセッサ・システムやマルチタスクがあります。

TST (テスト) は、オペランドに何の影響も与えることなく、ディスティネーションをゼロと比較します。そして、テスト結果によってコンディション・コードをセットします。

CLR (クリア) は、指定されたディスティネーションをバイト、ワード、ロング・ワードの全ビットをゼロにクリアします。

ADD.W D0,D1	データ・レジスタ D0 の下位ワードと D1 の下位ワードを加算する
SUB.B #3,(A0)	アドレス・レジスタ A0 の指示するアドレスの内容から“3”を引く
CMP.L \$1000,D0	\$1000 番地のロング・ワードをデータ・レジスタ D0 と比較し、コンディション・コードをセットする
BCLR #3,\$40(A0)	アドレス・レジスタ A0 の内容に \$40 を加えたアドレスのバイトのビット 3 をクリアする

図 7・6 整数算術演算命令の例

7 命 令 セ ッ ト

ADD: (ディスティネーション)+(ソース)→ディスティネーション

アセンブラ: ADD <ea>, Dn

ADD Dn, <ea>

	15	14	13	12	11		9	8		6	5		0
フォーマット	1	1	0	1	レジスタ		OP	モード		実効	アドレス		

ADDX: (ディスティネーション)+(ソース)+X→ディスティネーション

アセンブラ: ADDX Dy, Dx

ADDX- (Ay), -(Ax)

	15	14	13	12	11		9	8	7	6	5	4	3	2	0
フォーマット	1	1	0	1	Rx レジスタ		1	サイズ	00	R/M		Ry レジスタ			
					①			②		③		④			

① ディスティネーション・レジスタ番号

② オペレーション・サイズ

00	: バイト・オペレーション
01	: ワード・オペレーション
10	: ロング・ワード・オペレーション

③ オペランドのアドレッシング・モード

0	: データ・レジスタからデータ・レジスタ
1	: メモリからメモリ

④ ソース・レジスタ番号

DIVU: (ディスティネーション)/(ソース)→ディスティネーション

アセンブラ: DIVU <ea>, Dn

	15	14	13	12	11		9	8	7	6	5		0
フォーマット					レジスタ		0	1	1		実効	アドレス	

DIVS: (ディスティネーション)/(ソース)→ディスティネーション

アセンブラ: DIVS <ea>, Dn

	15	14	13	12	11		9	8	7	6	5		0
フォーマット	1	0	0	0	レジスタ		1	1	1		実効	アドレス	

MULU: (ディスティネーション)*(ソース)→ディスティネーション

アセンブラ: MULU <ea>, Dn

	15	14	13	12	11		9	8	7	6	5		0
フォーマット	1	1	0	0	レジスタ		0	1	1		実効	アドレス	

MULS: (ディスティネーション)*(ソース)→ディスティネーション

アセンブラ: MULS <ea>, Dn

	15	14	13	12	11		9	8	7	6	5		0
フォーマット	1	1	0	0	レジスタ		1	1	1		実効	アドレス	

CMP: (ディスティネーション)-(ソース)

アセンブラ: CMP <ea>, Dn

	15	14	13	12	11		9	8		6	5		0
フォーマット	1	0	1	1	レジスタ		OP	モード		実効	アドレス		

図 7.7 整数算術演算命令

7 命 令 セ ッ ト

SUB: (ディスティネーション)-(ソース)→ディスティネーション

アセンブラ: SUB <ea>, Dn

BUB Dn, <ea>

	15	14	13	12	11	9	8	6	5	0
フォーマット	1	0	0	1	レジスタ	OP モード				実効アドレス

SUBX: (ディスティネーション)-(ソース)-X→ディスティネーション

アセンブラ: SUBX Dy, Dx

SUBX-(Ay), -(Ax)

	15	14	13	12	11	9	8	7	6	5	4	3	2	0
フォーマット	1	0	0	1	Rx レジスタ	1	サイズ	0	0	R/M			Ry レジスタ	
					①		②			③			④	

① ディスティネーション・レジスタ番号

② オペレーション・サイズ

00:	バイト
01:	ワード
10:	ロング・ワード

③ オペランドのアドレッシング・モード

④ ソース・レジスタ番号

NEG: 0-(ディスティネーション)→ディスティネーション

アセンブラ: NEG <ea>

	15	14	13	12	11	10	9	8	7	6	5	0
フォーマット	0	1	0	0	0	1	0	0	サイズ			実効アドレス

NEGX: 0-(ディスティネーション)-X→ディスティネーション

アセンブラ: NEGX <ea>

	15	14	13	12	11	10	9	8	7	6	5	0
フォーマット	0	1	0	0	0	0	0	0	0	サイズ		実効アドレス

EXT: (ディスティネーション) 符号拡張→ディスティネーション

アセンブラ: EXT Dn

	15	14	13	12	11	10	9	8	6	5	4	3	2	0
フォーマット	0	1	0	0	1	0	0	OP モード	0	0	0			レジスタ

バイトからワードへ [EXT.W]



ワードからロング・ワードへ [EXT.L]



EXT 命令

図 7・8 (1) 整数算術演算命令

7 命 令 セ ッ ト

TAS: (ディスティネーション) テスト→CC

1 → ディスティネーションの #7

アセンブラ: TAS <ea>

TST: (ディスティネーション) テスト→CC

アセンブラ: TST <ea>

	15	14	13	12	11	10	9	8	7	6	5		0
フォーマット	0	1	0	0	1	0	1	0	サイズ	実効アドレス			

CLR: 0 → ディスティネーション

アセンブラ: CLR <ea>

	15	14	13	12	11	10	9	8	7	6	5		0
フォーマット	0	1	0	0	0	0	1	0	サイズ	実効アドレス			

図 7・8 (2) 整数算術演算命令

MC 68000 外観



7.4 論理操作命令

表7.5 に示す論理操作命令は、バイト、ワード、ロング・ワードのいずれのオペランドにも用いることができます。このほかにANDI（アドレス・イミディエート）、DRI（オア・イミディエート）、EORI（エクスクルーシブ・オア・イミディエート）などがあります。

AND（アンド）命令は、ソース・オペランドとディスティネーション・オペランドとANDをとり、ディスティネーションに収納します。

OR（オア）命令は、ソース・オペランドとディスティネーション・オペランドとインクルーシブORをとり、ディスティネーションに収納します。

EOR（エクスクルーシブ・オア）命令は、ソース・オペランドとディスティネーション・オペランドとEORをとり、ディスティネーションに収納します。

NOT（ノット）命令は、ディスティネーション・オペランドの“1”の補数を取り、ディスティネーションに収納します。

表 7.5 論理操作命令

命 令	オペランド・サイズ	機 能	内 容
AND	8, 16, 32	$D_n \wedge (EA) \rightarrow D_n$ $(EA) \wedge D_n \rightarrow EA$ $(EA) \wedge \#xxx \rightarrow EA$	論理AND
OR	8, 16, 32	$D_n \vee (EA) \rightarrow D_n$ $(EA) \vee D_n \rightarrow EA$ $(EA) \vee \#xxx \rightarrow EA$	論理インクルーシブOR
EOR	8, 16, 32	$(EA) \oplus D_y \rightarrow EA$ $(EA) \oplus \#xxx \rightarrow EA$	論理EOR
NOT	8, 16, 32	$\sim (EA) \rightarrow EA$	論理否定

～：インバート

7 命 令 セ ッ ト

AND: (ディスティネーション) ∧ (ソース) → ディスティネーション

アセンブラ: AND <ea>, Dn

AND Dn, <ea>

	15	14	13	12	11	9	8	6	5	0
フォーマット	1	1	0	0	レジスタ	OP モード	実効アドレス			
	①				②		③			

① レジスタ番号

② $\begin{cases} 000: \text{バイト} & 001: \text{ワード} & 010: \text{ロング・ワード} \\ 100: & 101: & 110: \end{cases}$

③ $\begin{cases} \text{ソース・オペランド時} \\ \text{データ・アドレッシング・モードのみ可} \\ \text{ディスティネーション・オペランド時} \\ \text{可変とメモリのアドレッシング・モードのみ可} \end{cases}$

OR: (ディスティネーション) ∨ (ソース) → ディスティネーション

アセンブラ: OR <ea>, Dn

OR Dn, <ea>

	15	14	13	12	11	9	8	6	5	0
フォーマット	1	0	0	0	レジスタ	OP モード	実効アドレス			
	①				②		③			

① 8データ・レジスタ番号

② $\begin{cases} 000: \text{バイト} & 001: \text{ワード} & 010: \text{ロング・ワード} \\ 100: & 101: & 110: \end{cases}$

③ $\begin{cases} \text{ソース・オペランド時} \\ \text{データ・アドレッシング・モードのみ可} \\ \text{ディスティネーション・オペランド時} \\ \text{可変とメモリ・アドレッシングのみ可} \end{cases}$

EOR: (ディスティネーション) ⊕ (ソース) → ディスティネーション

アセンブラ: EOR Dn, <ea>

	15	14	13	12	11	9	8	6	5	0
フォーマット	1	0	1	1	レジスタ	OP モード	実効アドレス			
	①				②		③			

① データ・レジスタ番号

② $\begin{cases} 100: \text{バイト} \\ 101: \text{ワード} \\ 110: \text{ロング・ワード} \end{cases}$

③ データと可変のアドレッシング・モードのみ可

NOT: ~(ディスティネーション) → ディスティネーション

アセンブラ: NOT <ea>

	15	14	13	12	11	10	9	8	7	6	5	0
フォーマット	0	1	0	0	0	1	1	0	サイズ	実効アドレス		

① データと可変のアドレッシング・モードのみ可









図 7.9 論理操作命令

7.5 シフトとローテット命令

シフト命令には、LSL, LSR の二つの符号なし論理シフト命令と、二つの ASL, ASR の符号付き算術命令の4種類があり、ローテット命令には、ROR, ROL の二つの拡張なしローテット命令と、二つの ROXL, ROXR の拡張付きローテット命令の4種類があります。またオペランドは、左右方向シフトと左右方向のローテットが可能です。表7.6に示すように、各シフト命令の機能を有効に利用することにより、乗算命令や除算命令より効率の良い演算の実行が可能です。

ASL, ASR (アリスメティック・シフト・レフトとライト) は、ディスティネーション・オペランドのビットを指定方向に算術シフトします。シフトによって送り出されたビットは、キャリー・ビット (C) と拡張ビット (x) にそれぞれ入

表 7.6 シフトおよびローテット命令

命 令	オペランド・サイズ	機 能	内 容
ASL	8, 16, 32		符号付き算術シフト
ASR	8, 16, 32		
LSL	8, 16, 32		符号なし論理シフト
LSR	8, 16, 32		
ROL	8, 16, 32		拡張なしローテット
ROR	8, 16, 32		
ROXL	8, 16, 32		拡張付きローテット
ROXR	8, 16, 32		

7 命 令 セ ッ ト

ります。ASL 命令では、最下位ビットに 0 が入り、ASR 命令では符号ビットが最上位ビットに再度入れられます。

LSL, LSR (ロジカル・シフト・レフトとライト) は、ディスティネーション・オペランドのビットを指定方向に論理シフトします。シフトによって送り出されたビットは、キャリー・ビット (C) と拡張ビット (x) にそれぞれ入ります。LSL 命令では、最下位ビットに 0 が入り、LSR 命令では最上位ビットに 0 が入ります。

ROL, ROR (ローティット・ウイズアウト・エセステンド・レフトとライト) は、ディスティネーション・オペランドのビットを指定方向にローテートさせます。なお、拡張ビットはローテートされません。ROL 命令では、最上位から送出されたビットは、キャリー・ビットと最下位ビットの両方に入り、ROR 命令では最下位から送出されたビットはキャリー・ビットと最上位ビットの両方に入ります。

ROXL, ROXR (ローティット・ウイズ・エクステンド・レフトとライト) は、ディスティネーション・オペランドのビットを指定方向にローテートさせます。また、拡張ビットがローティットに含まれます。ROXL は、最上位ビットから送り出されたビットはキャリーと拡張の両ビットに入り、拡張ビットの前の値が最下位ビットに送り込まれます。ROXR は、最下位ビットから送り出されたビットはキャリーと拡張の両ビットに入り、拡張ビットの前の値が最上位ビットに送り込まれます。

例

LSL.W #3, D0	データ・レジスタ D0 の下位ワードを左に 3 ビット、ロジカル・シフトする。
ASR #2, (A2)	アドレスレジスタ A2 の指定するアドレスのワードを 2 ビット右にアリスメティック・シフトする。
ROXL.B D2, D1	データレジスタ D1 の下位バイトを左に、D2 の内容に従いローテーション・シフトする。

7 命 令 セ ッ ト

ASL : (ディスティネーション) <カウント> によるシフト→ディスティネーション
ASR :

アセンブラ : ASd Dx, Dy

ASd # <データ>, Dy

ASd <ea>

フォーマット
(レジスタ・
シフト)

15	14	13	12	11	9	8	7	6	5	4	3	2	0
1	1	1	0	カウント/ レジスタ	dr	サイズ	i/r	00	レジスタ				

(メモリ・
シフト)

15	14	13	12	11	10	9	8	7	6	5	0
1	1	1	0	0	0	0	dr	1	1	実効アドレス	

①

②

① シフト方向 0 : 右シフト
1 : 左シフト

② メモリと可変アドレッシング・モードのみ可

LSR : (ディスティネーション) <カウント> によるシフト→ディスティネーション
LSL :

アセンブラ : LSd Dx, Dy

LSd # <データ>, Dy

LSd <ea>

フォーマット
(レジスタ・
シフト)

15	14	13	12	11	9	8	7	6	5	4	3	2	0
1	1	1	0	カウント/ レジスタ	dr	サイズ	i/r	0	1	レジスタ			

①

②

③

① 00 : バイト
01 : ワード
10 : ロング・ワード

② 0 : シフト回数をイミディエート・データで指定
1 : シフト回数をデータ・レジスタで指定

③ シフトされるデータ・レジスタの番号

15	13	12	10	9	8	7	6	5	0	
1	1	1	0	0	0	1	dr	1	1	実効アドレス

①

②

① シフト方向 0 : 右シフト
1 : 左シフト

② メモリと可変アドレッシング・モードのみ可

図 7・10 (1)

7 命令セット

ROL: (ディスティネーション) 〈カウント〉 によるローテート→ディスティネーション

アセンブラ: R0d Dx, Dy

ROd # 〈データ〉, Dy

ROd <ea>

フォーマット
(レジスタ・
ローテート)

15	13	12	11	9	8	7	6	5	4	3	2	0
1	1	1	0	カウント/ レジスタ	dr	サイズ	i/r	1	1	レジスタ		

① ②

- ① 〔0:ローテート回数をイミディエート・データで指示
1:ローテート回数をデータ・レジスタで指示

② ローテートされるデータ・レジスタの番号

(メモリ・
日一テート)

15	13	12	11	10	9	8	7	6	5	0
1	1	1	0	0	1	1	dr	1	1	実効アドレス

① ②

- ① $\begin{cases} 0: \text{右ローテート} \\ 1: \text{左ローテート} \end{cases}$

② メモリと可変のアドレッシングのみ可

ROXL: (ディスティネーション)〈カウント〉によるローテート→ディスティネーション

アセンブラ: ROXd Dx, Dy

ROXd #〈データ〉, Dy

ROXd <ea>

フォーマット
(レジスタ・
ローテート)

15	13	12	11	9	8	7	6	5	4	3	2	0
1	1	1	0	カウント/ レジスタ	dr	サイズ	i/r	1	0	レジスタ		

① ② ③

- ① $\begin{cases} 0: \text{右ローテート} \\ 1: \text{左ローテート} \end{cases}$

- ② [0 : ロータート回数をイミディエート・データで指定
1 : ロータート回数をデータ・レジスタで指定

③ ローテートされるデータ・レジスタ番号

(メモリ・
ローテート)

15	13	12	11	10	9	8	7	6	5	0
1	1	1	0	0	1	0	dr	1	1	実効アドレス

① ②

- ① $\begin{cases} 0: \text{右ローテイト} \\ 1: \text{左ローテイト} \end{cases}$

② メモリと可変のアドレッシング・モードのみ可

图 7·10 (2)

7.6 BCD 演算命令

BCD 演算命令には、ABCD（拡張 10 進加算）命令、SBCD（拡張 10 進減算）命令、NBCD（拡張 10 進ネグート）命令の三つがあり、2 進化 10 進数（BCD）をバイト単位で扱う命令です。この三つの演算命令は、常に拡張フラグ（x）が含まれますので、プログラムするうえで X フラグの配慮が必要です。

ABCD（アド・デシマル・エクステンド）は、ソース・オペランドを拡張ビット（x）とともにディスティネーション・オペランドに加えて、ディスティネーションに収納します。

SBCD（サブトラクト・デシマル・エクステンド）は、ソース・オペランドと拡張ビット（x）をディスティネーション・オペランドから BCD 演算を用いて減算して、ディスティネーションに収納します。

NBCD（ネグート・デシマル・エクステンド）は、ゼロからディスティネーション・オペランドと拡張ビットを減算して、ディスティネーションに収納します。オペレーションは 10 進演算を用います。

表 7.7 BCD 演算命令

命 令	オペランド・サイズ	機 能	内 容
ABCD	8	$Dx10 + Dy10 + X \rightarrow Dx$ $Ax@-10 + Ay@-10 + X \rightarrow Ax@$	拡張付き 10 進加算
SBCD	8	$Dx10 - Dy10 - X \rightarrow Dx$ $Ax@-10 - Ay@-10 - X \rightarrow Ax@$	拡張 10 進減算
NBCD	8	$0 - (EA)10 - X \rightarrow EA$	拡張付き 10 進補数

7 命 令 セ ッ ト

ABCD: (ディスティネーション)₁₀ + (ソース)₁₀ + X → ディスティネーション

アセンブラ: ABCD Dy, Dx

ABCD - (Ay), -(Ax)

	15	14	13	12	11	9	8	7		4	3	2	0
フォーマット	1	1	0	0	Rx レジスタ	1	0	0	0	0	R/M	Ry レジスタ	
					①						②	③	

① ディスティネーションのレジスタ番号

② オペランドのアドレッシング・モード指定

0: データ・レジスタからデータ・レジスタへのオペレーション
1: メモリからメモリへのオペレーション

③ ソース・レジスタ番号

SBCD: (ディスティネーション)₁₀ - (ソース)₁₀ - X → ディスティネーション

アセンブラ: SBCD Dy, Dx

SBCD - (Ay), -(Ax)

	15	14	12	11	9	8	7		4	3	2	0
フォーマット	1	0	0	0	Rx レジスタ	1	0	0	0	0	R/M	Ry レジスタ
					①						②	③

① ディスティネーションのレジスタ番号

② オペランドのアドレッシング・モードを指定

0: データ・レジスタからデータ・レジスタ
1: メモリからメモリ

③ ソース・レジスタ番号

NBCD: 0 - (ディスティネーション)₁₀ - X → ディスティネーション

アセンブラ: NBCD <ea>

	15	14	13	12	11	10		6	5		0
フォーマット	0	1	0	0	1	0	0	0	0	0	実効アドレス
											①

① データと可変のアドレッシング・モードのみ可

図 7-11 BCD 演算命令

7.7 ビット操作命令

ビット操作命令には、表 7.8 に示すように **BTST** (ビット・テスト), **BSET** (ビット・テストとセット), **BCLR** (ビット・テストとクリア), **BCHG** (ビット・テストと変更) の 4 種類の命令があります。これらはいずれもバイトかロング・ワードのオペランドを用いて、ビット・テストの結果に応じてビット操作を行います。

BTST (テスト・ビット) は、ディスティネーションのうちのソース・オペランドで示されるビットの状態を調べ、そのビットの状態によりコンディション・コードのゼロ・フラグ (**Z**) をセットします。

BSET (テスト・ビット・アンド・セット) は、**BTST** と同じ方法でビット・テストを行い、テストの後にテスト・ビットがディスティネーションによってセットされます。

BCLR (テスト・ビット・アンド・クリア) は、**BTST** と同じ方法でビット・テストを行い、テストの後にテスト・ビットはディスティネーションによってクリアされます。

BCHG (テスト・ビット・アンド・チェンジ) は、**BTST** と同じ方法でビット・テストを行い、テストの後にテスト・ビットはディスティネーションによって反転されます。

例

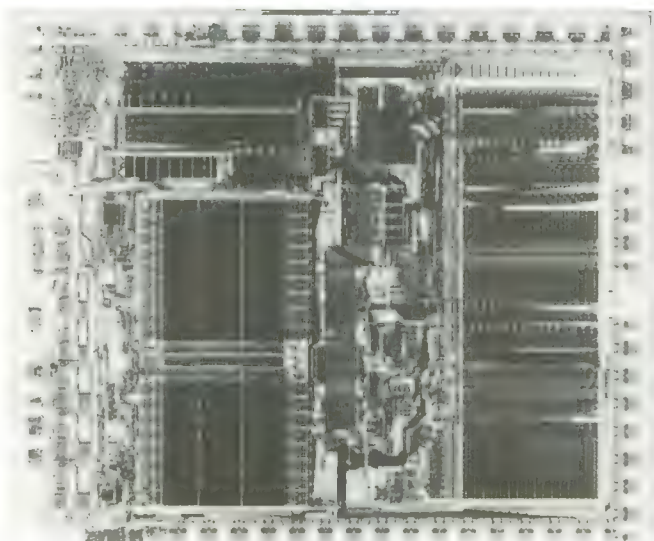
BCLR #2, \$40 (A3) BCHG D1, D2	アドレス・レジスタ A3 の内容に \$40 を加えたアドレスのバイトのビット 2 をクリアする。 データ・レジスタ D1 で指示されたビット番号で D2 のビットがチェックされる。チェック結果はコンディション・コードの Z ビットに表示され、 D2 の指定されたビットは反転される。
---	---

7 命 令 セ ッ ト

表 7・8 ビット操作命令

命 令	オペランド・サイズ	機 能	内 容
BTST	8, 32	$\sim(EA)$ のビット $\rightarrow Z$	ビット・テスト
BSET	8, 32	$\sim(EA)$ のビット $\rightarrow Z$ 1 $\rightarrow EA$ のビット	ビット・テストとセット
BCLR	8, 32	$\sim(EA)$ のビット $\rightarrow Z$ 0 $\rightarrow EA$ のビット	ビット・テストとクリア
BCHG	8, 32	$\sim(EA)$ のビット $\rightarrow Z$ $\sim(EA)$ のビット $\rightarrow EA$ のビット	ビット・テストと変更

MC 68000 内部拡大図



7 命 令 セ ッ ト

BTST : ~(ディスティネーションの〈ビット番号〉)→Z

アセンブラ : BTST Dn, <ea>

BTST # <データ>, <ea>

フォーマット (ビット番号) ダイナミック	15	12	11	9	8	7	6	5	0
	0	0	0	0	レジスタ	1	0	0	実効アドレス
	①				②				

① ビット番号を格納しているデータ・レジスタの番号

② アドレッシング・カテゴリのデータと可変のアドレッシング・モードのみ可

	15	12	11	10	6	5	①	0		
(ビット番号) (スタティック)	0	0	0	0	1	0	0	0	0	実効アドレス
	ビット番号									

① イミディエート・アドレッシング・モードを除いたアドレッシング・カテゴリのデータ可変のアドレッシング・モードのみ可

BSET : ~(ディスティネーションの〈ビット番号〉)→Z

アセンブラ : BSET Dn, <ea>

BSET # <データ>, <ea>

	15	12	11	10	8	7	6	5	①	0	
フォーマット (ビット番号) (スタディック)	0	0	0	0	1	0	0	0	1	1	実効アドレス
	ビット番号										

① データと可変のアドレッシング・モードのみ可

BCLR : ~(ディスティネーションの〈ビット番号〉)→Z

アセンブラ : BCLR Dn, <ea>

BCLR # <データ>, <ea>

フォーマット (ビット番号) ダイナミック	15	12	11	9	8	7	6	5	0
	0	0	0	0	レジスタ	1	1	0	実効アドレス

(ビット番号) (スタティック)	15	12	11	10	8	7	6	5	0		
	0	0	0	0	1	0	0	0	1	0	実効アドレス
	ビット番号										

BCHG : ~(ディスティネーションの〈ビット番号〉)→Z

アセンブラ : BCHG Dn, <ea>

BCHG # <データ>, <ea>

フォーマット (ビット番号) ダイナミック	15	12	11	9	8	7	6	5	0
	0	0	0	0	レジスタ	1	0	1	実効アドレス

	15	14	13	12	11	10	9	8	7	6	5	0
(ビット番号) (スタティック)	0	0	0	0	1	0	0	0	0	1	実効アドレス	
	0	0	0	0	0	0	0	0	ビット番号			

図 7・12

7・8 プログラム制御命令

プログラムの構成はジャンプやブランチを伴い、サブルーチンによるコール・プログラム・ルーチンなど、必ずしも順序正しくプログラムが実行されるとは限りません。このようなプログラムの制御命令には、表7・9に示すように条件付き、無条件、リターンと三つの分類ができます。条件付き命令にはBcc, DBcc, Sccの3種類があり、表7・10に示す16種のテスト・コンディションを備え、コンディ

表 7・9 プログラム制御命令

区 分	命 令	機 能
条件付き	Bcc DBcc Scc	条件付きブランチ (14条件) 8ビットと16ビット変位 テスト条件、デクリメント・カウンタとブランチ 16ビット変位 バイト条件セット (16条件)
無 条 件	BRA BSR JMP JSR	無条件ブランチ 8ビットと16ビット変位 サブルーチンヘジャンプ 8ビットと16ビット変位 ジャンプ サブルーチンヘジャンプ
リターン	RTR RTS	リターンと条件コード復元 サブルーチンに戻り

表 7・10 テスト・コンディション

ニーモニック	コンディション	エンコーディング	テ ス ト
T	トルー	0000	1
F	フェール	0001	0
HI	ハイ	0010	$\bar{C} \cdot \bar{Z}$
LS	ローカセーム	0011	$C + Z$
OC	キャリー・クリア	0100	\bar{C}
CS	キャリー・セット	0101	C
NE	ノット・イコール	0110	\bar{Z}
EQ	イコール	0111	Z
VC	オーバフロー・クリア	1000	\bar{V}
VS	オーバフロー・セット	1001	V
PL	プラス	1010	\bar{N}
MI	マイナス	1011	N
GE	大きいか等しい	1100	$N \cdot V + \bar{N} \cdot \bar{V}$
LT	小さい	1101	$N \cdot \bar{V} + \bar{N} \cdot V$
GT	大きい	1110	$N \cdot V \cdot \bar{Z} + \bar{N} \cdot \bar{V} \cdot \bar{Z}$
LE	小さいか等しい	1111	$Z + N \cdot \bar{V} + \bar{N} \cdot V$

ション付きブランチ命令とセット命令のコンディション・コードとテストの関係を示します。無条件命令にはBRA, BSR, JMP, JSR の4種類があり、リターン命令にはRTR, RTS の2種類があります。

Bcc 命令は、指定されたテスト条件が満足すると、プログラムの実行はプログラム・カウンタ (PC)+変位にブランチします。

DBcc 命令は、コンディション、データレジスタ、ディスティネーションの三つのパラメータによる繰返しループ命令のターミネータとして働きます。

① 指定した条件のコンディションをテストし、満足されたかどうかを判定します。

② 条件が満たされているときは、ブランチせず、次の命令を実行します。

③ 条件が満たされない場合は、レジスタの値を“1”だけ減算します。

④ もしレジスタの値が“-1”ならばブランチせず、次の命令を実行します。

⑤ -1 でないときは、PC の値に符号拡張された16ビット変位を加えた位置から続行されます。

Scc 命令は、指定したコンディションを調べ、条件が満たされているときは、実効アドレスで指定されるバイトがすべて1にセットされ、条件が満たされない場合には、すべて0にセットされます。

BRA 命令におけるプログラム実行は、プログラム・カウンタ (PC)+変位 (d) で続行されます。

BSR 命令の直後の命令のアドレスはシステム・スタックに収納され、その後、プログラムの実行は変位の位置で続行されます。

JMP 命令におけるプログラム実行は、命令で指定される制御アドレッシング・モードで指定されるアドレスで続行されます。

JSR 命令の直後のアドレスがシステム・スタックに収納され、その後、プログラム実行は命令で指定されたアドレスで続行します。

RTR, RTS 命令は、コンディション・コードとPCをスタックから取り出します。また、命令実行前のコンディション・コードとPCは消去されます。

7 命 令 セ ッ ト

Bcc: もし cc ならば $PC+d \rightarrow PC$

アセンブラ: Bcc <ラベル>

	15	14	13	12	11	8	7	0
フォーマット	0	1	1	0	条件	8ビット変位		
	8ビット変位=0 のとき 16ビット変位							

DBcc: もし cc でなければ $Dn-1 \rightarrow Dn$

もし $Dn \neq 1$ ならば $PC+d \rightarrow PC$

アセンブラ: DBcc Dn, <ラベル>

	15	14	13	12	11		8	7	6	5	4	3	2	0
フォーマット	0	1	0	1	条件		1	1	0	0	1	レジスタ		
	変 位													

Sc: もし cc ならば “1” → ディスティネーション

もし cc でない “0” → ディスティネーション

アセンブラ: Sc <ea>

15	14	13	12	11	8	7	6	5	0
0	1	0	1	条件	1	1	実効アドレス		

BRA: $PC+d \rightarrow PC$

アセンブラ: BRA <ラベル>

15	14	13	12	11	10	9	8	7	0
0	1	1	0	0	0	0	0	8ビット変位	
8ビット変位=0のとき 16ビット変位									

BSR: $PC \rightarrow SP@-; PC-d \rightarrow PC$

アセンブラ: BSR <ラベル>

15	14	13	12	11	10	9	8	7	0
0	1	1	0	0	0	0	1	8ビット変位	
8ビット変位=0のとき 16ビット変位									

JMP: ディスティネーション → PC

アセンブラ: JMP <ea>

15	14	13	12	11	10	9	8	7	6	5	0
0	1	0	0	1	1	1	0	1	1	実効アドレス	

①

① ジャンプ先のアドレス指定, 制御アドレッシング・モードのみ可

JSR: $PC \rightarrow SP@-; ディスティネーション \rightarrow PC$

アセンブラ: JSR <ea>

15	14	13	12	11	10	9	8	7	6	5	0
0	1	0	0	1	1	1	0	1	0	実効アドレス	

①

① ジャンプ先のアドレス指定, 制御アドレッシング・モードのみ可

7 命 令 セ ッ ト

RTR: SP@+ → CC; SP@+ → PC

アセンブラ: RTR

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
フォーマット	0	1	0	0	1	1	1	0	0	1	1	1	0	1	1	1

RTS: SP@+ → PC

アセンブラ: RTS

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
フォーマット	0	1	0	0	1	1	1	0	0	1	1	1	0	1	0	1

図 7・13 プログラム制御命令

```
*      SCC INSTRUCTION EXAMPLE
*
*THE SCC INSTRUCTION ALLOWS THE USER
*TO REMEMBER THE RESULT OF AN INSTRUCTION
*AND ACT UPON IT AT SOME LATER TIME
*
ORG $2000
CLR D0
CLR D1
CMP D0,D1
SEQ D2
ADDQ #1,D0
CMP D0,D1
SEQ D2
END
```

図 7・14 SCC プログラムの例

7 命 令 セ ッ ト

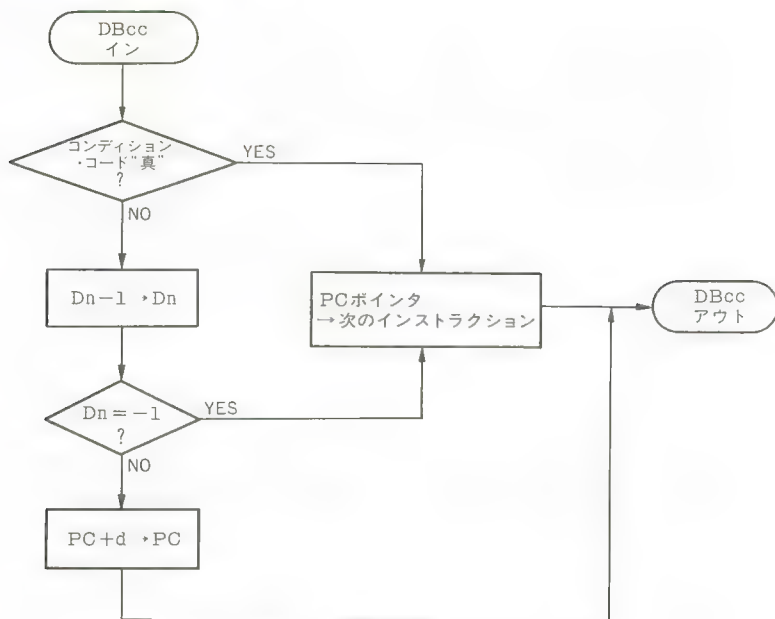


図 7・15 DBcc 命令フロー図

条件付きジャンプ

DBEQ DO, \$1000

コンディションが“EQ”ならば次の命令を実行し，“EQ”でないならば“DO”から“1”引く。そして、引かれた結果がマイナスならば次の命令を実行する。マイナスでなければ\$1000番地に飛ぶ

BNE.L \$5000

コンディションが“NE”のとき\$5000番地へロング・ジャンプする

BEQ.S \$5000

コンディションが“EQ”のとき\$5000番地へショート・ジャンプする

無条件ジャンプ

JMP 2(A7)

A7に2を加えたアドレスへ飛ぶ

JMP.L \$1000

\$1000番地へ飛ぶ。アブソリュート・ロング・ジャンプ

JSR \$5000

\$5000番地のサブルーチンへ飛ぶ

BSR \$5000

\$5000番地にあるサブルーチンへ飛ぶ

図 7・16 プログラム制御命令の例

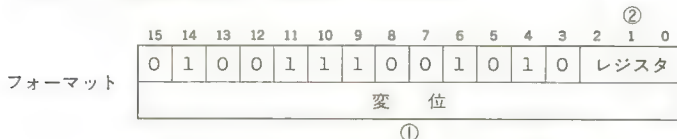
7.9 リンク命令とアンリンク命令

リンク命令とアンリンク命令は、メモリ・スタックの割当てや復帰のための手順やサブルーチン用法などの命令です。そして、有効なデータ領域への使用や、作業領域の確保としてのシステム・スタック用法でもあります。また、プログラミングするうえで非常に有益な命令であり、リンク命令によってスタック・ポインタをどのように変更しても、UNLK 命令を使用することによりフレーム・ポインタやスタック・ポインタがLINK 命令前の値に自動的に復帰されることは大きな魅力といえます。

LINK (リンク) 命令は、指定されたアドレス・レジスタの内容がスタックに収納され、その後、プリデクリメント (-4) されたスタック・ポインタの値がアドレス・レジスタにロードされます。そして、符号拡張されたディスプレイメントがスタック・ポインタに加えられます。

LINK: $An \rightarrow SP@-; SP \rightarrow An; SP+d \rightarrow SP$

アセンブラ: LINK $An, \# \langle \text{ディスプレイメント} \rangle$



① スタック・ポインタに加算する“2”の補数の整数

② リンクするためのアドレス・レジスタの番号

図 7-17

UNLK (アンリンク) 命令は、指定されたアドレス・レジスタがスタック・ポインタにロードされ、アドレス・レジスタへスタックの先頭から取り出されたロング・ワードがロードされます。

UNLK: $An \rightarrow SP; SP@+ \rightarrow An$

アセンブラ: UNLK An



① アドレス・レジスタ番号

図 7-18

7 命 令 セ ッ ト

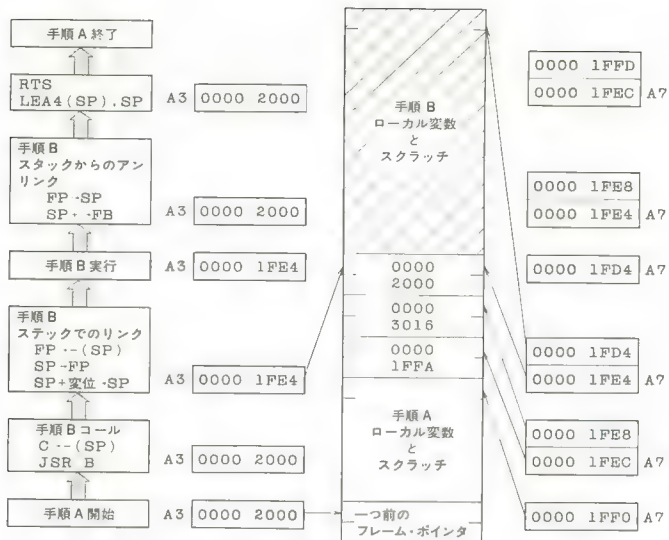


図 7-19 プログラム・フロー図

LINKUN MC68000 ASM

```

1      00003000      ORG $3000
2      003000 47F82000      LEA $2000,A3
3      003004 4FF81FF0      LEA $1FF0,SP
4      003008 4E71      PROC A  NOP ----- ラベル "PROCA"
5      00300A 4E71      NOP
6      00300C 4E71      NOP
7      00300E 486BFFFA      PEA -6(A3)
8      003012 4EB83022      JSR PROCB ----- リンク・ルーチンへジャンプ
9      003016 4FEF0004      LEA 4(SP),SP
10     00301A 4E71      NOP
11     00301C 4E71      NOP
12     00301E 4E71      NOP
13     003020 60FE      ENDA BRA * ----- ラベル "ENDA"
14     003022 4E53FFFO      PROCB LINK A3,-$10 ----- ラベル "PROCB"
15     003026 4E71      NOP
16     003028 4E71      NOP
17     00302A 4E71      NOP
18     00302C 4E5B      UNLK A3
19     00302E 4E75      RTS ----- リターン
20     END

```

***** TOTAL ERRORS 0-- 0

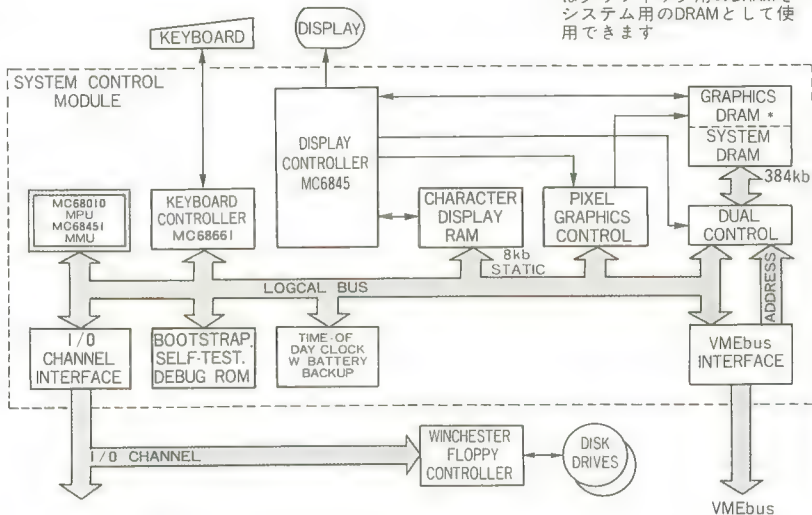
SYMBOL TABLE

ENDA 003020 PROC A 003008 PROC B 003022

図 7-20 リンク、アンリンクのプログラム例

■ VME/10 システムブロック図

VME 10システム・ブロック図



特 長

1. MC 68010 16ビット仮想マシン搭載。
2. MC 68451 メモリ管理用 LSI 搭載。
3. VME bus バス仕様と I/O Channel インタフェース。
4. 32K バイトのダイナミック RAM と 8K バイトのスタティック RAM 実装。
5. 表示形式をソフトウェアで制御できる。
6. フロッピー・ディスク (1M バイトのアンフォーマット)。
7. ウィンチェスタ・ディスク (6M/19M バイトのアンフォーマット)。

VME/10 マイクロコンピュータ・システム (その2) <その3は p.143>

7.10 システム制御命令

システム制御命令は、表 7・11 に示すように特権命令、トラップ命令、ステータス・レジスタ命令と機能別に 3 タイプに分類できます。

RESET (リセット) は、リセット・ラインをセットし、すべての外部デバイスをリセットしますが、68000 の内部レジスタはプログラム・カウンタ以外のレジスタ群は変化しません。

RTE (リターン・フロム・エクセプション) は、スタックからプログラム・カウンタとステータス・レジスタを読み出し、プログラム・カウンタの示すアドレスから処理を続行します。

STOP (ロード・ステータス・レジスタ・アンド・ストップ) は、イミディエート・データの 16 ビットがステータス・レジスタに転送され、68000 は命令フェッチと実行を停止します。

MOVE USP (ムーブ・ユーザ・スタック・ポインタ) は、ユーザ・スタック・ポインタの内容と指定されたアドレス・レジスタとを転送します。

表 7・11 システム制御命令

区 分	命 令	機 能
特 権 命 令	RESET RTE STOP ORI to SR MOVE USP ANDI to SR EORI to SR MOVE EA to SR	外部デバイス・リセット 例外からのリターン プログラム例外のストップ ステータス・レジスタへの論理 OR ユーザ・スタック・ポインタへの転送 ステータス・レジスタへの論理 AND ステータス・レジスタへの論理 EOR 新しいステータス・レジスタのロード
トラップ命令	TRAP TRAPV CHK	トラップ オーバフロー・トラップ レジスタ限界チェック
ステータス・レジスタ	ANDI to CCR EORI to CCR MOVE EA to CCR ORI to CCR MOVE SR to EA	コンディション・コードへの論理 AND コンディション・コードへの論理 EOR 新しいコンディション・コードのロード コンディション・コードへの論理 OR ステータス・レジスタの収納

7 命 令 セ ッ ト

TRAP (トラップ) は、68000 が無条件で例外処理を開始します。命令の下位 4 ビットで指定されるトラップ命令の例外ベクタを参照するようベクタ番号が作られます。

TRAPV (トラップ・オン・オーバフロー) は、68000 がオーバフロー状態のときに例外処理を開始します。

CHK (チェック・レジスタ・アゲンスト・ボウダ) は、指定したデータ・レジスタの下位ワードの値とソース・オペランドで示される“2”の補数の整数と比較し、レジスタ値が 0 より小さいか、指定した整数値より大きいと、68000 は例外処理を開始します。

ステータス・レジスタ命令は、コンディション・コードに対する論理 AND、論理 EOR、論理 OR のオペレーションや、コンディション・コードのセッティングやレジスタへの収納などの命令です。

RESET:

アセンブラ: RESET

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
フォーマット	0	1	0	0	1	1	1	0	0	1	1	1	0	0	0	0

RTE: SP@+→SR

SR@+→PC

アセンブラ: RTE

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
フォーマット	0	1	0	0	1	1	1	0	0	1	1	1	0	0	1	1

STOP: イミディエート・データ→SR

アセンブラ: STOP #xxx

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
フォーマット	0	1	0	0	1	1	1	0	0	1	1	1	0	0	1	1
イミディエート・データ																

①

① SR にロードするイミディエート・データを指定

図 7・21 (1)

7 命 令 セ ッ ト

MOVE USP: USP → An

An → USP

アセンブラ: MOVE USP, An

MOVE An, USP

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
フォーマット	0	1	0	0	1	1	1	0	0	1	1	0	dr			レジスタ

① ②

① [アドレス・レジスタをユーザ・スタック・ポインタへ転送: 0]
 ユーザ・スタック・ポインタをアドレス・レジスタへ転送: 1]

② アドレス・レジスタ番号

TRAP: PC → SSP@-

SR → SSP@-

(ベクタ) → PC

アセンブラ: TRAP # <ベクタ>

	15	14	13	12	11	10	9	8	7	6	5	4	3			0
フォーマット	0	1	0	0	1	1	1	0	0	1	0	0			ベクタ	

①

① 16個のベクタ番号の指定

TRAPV: もし V なら TRAP

アセンブラ: TRAPV

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
フォーマット	0	1	0	0	1	1	1	0	0	1	1	1	0	1	1	0

CHK: もし Dn <0 か Dn> (<ea>) なら TRAP

アセンブラ: CHK <ea>, Dn

	15	14	13	12	11		9	8	7	6	5					0
フォーマット	0	1	0	0		レジスタ		1	1	0					実効アドレス	

①

②

① データ・レジスタ番号. この内容が調べられる

② ソース・オペランドを指定し, この内容が上限値.
 データ・アドレッシング・モードのみ可

図 7・21 (2)

8. システムの構成

68000 のシステム構成を，基本的なシステム構成をベースに，最小システム構成の条件や 68000 ファミリ・チップを用いた中位システム構成，上位システム構成へと展開し，またシステムと VE RSAbus，モジュールの関係を説明します。

8・1 最小システム

68000 における最小システム構成は、図 8・2 に示すような四つの基本的機能が要求されます。これらの基本構成に基づいて構成されたシステムが、図 8・1 (a) と (b) です。メモリや 6800 I/O の仕様は、MC68000 をどのようなスピードのものを使用するか、また、どのような I/O を使用するかによって分かれてきます。

例えば、メモリ関係は **MC6810** スタティック RAM、**MC6829** MMU があり、DMA コントローラには **MC6844** (並列転送)、**MC6855** (直列転送) があり、並列 I/O インタフェースには **MC6821** (PIA)、直列インタフェースには **MC6850** (非同期型)、**MC6852** (同期型)、**MC6854** (データ・リンク型) があり、ディスプレイ・コントローラには **MC6845** (CRT コントローラ)、**MC6847** (ディスプレイ・コントローラ) などがあります。

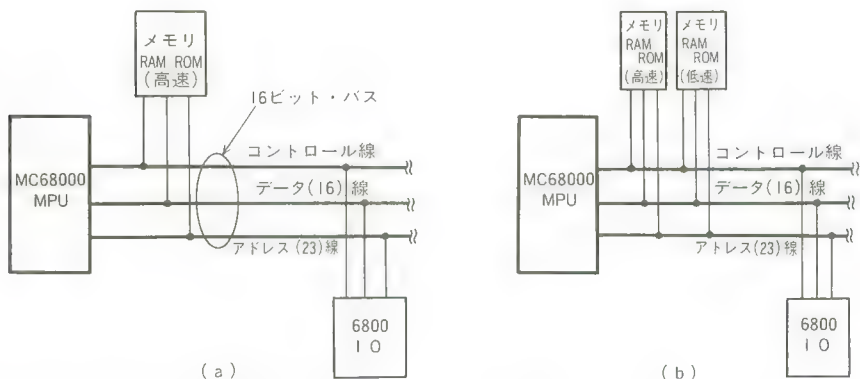


図 8・1 68000 最小システム

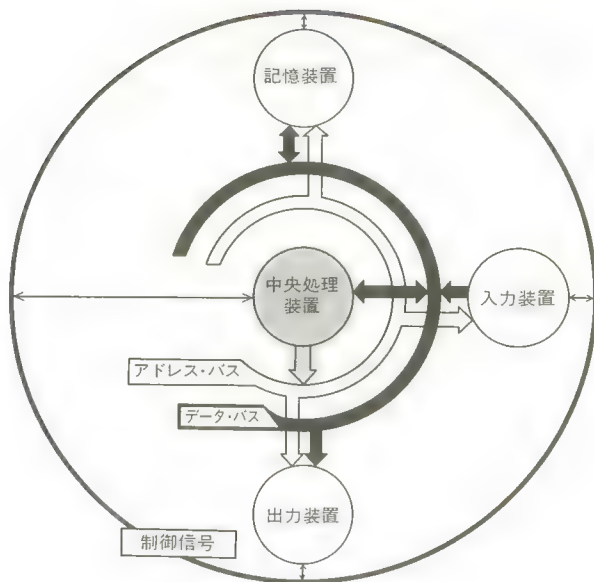


図 8・2 システムの基本構成

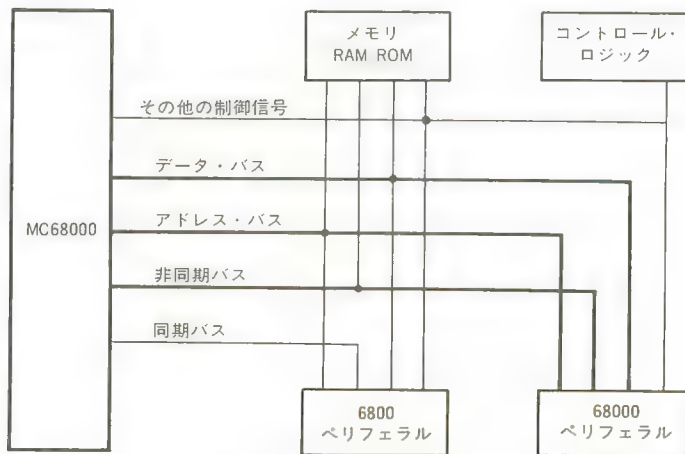


図 8・3 68000の基本システム

8.2 中位システム構成

中位システム構成は、図 8・4 に示すように、最小システムに **MC68120 IPC**（インテリジェント・ペリフェラル・コントローラ）が数個追加されることにより、中規模システムの構成となります。

数個の IPC が追加されることにより、数個のマイクロプロセッサやメモリ（RAM, ROM）や I/O ポートを備えることができ、最小システムに比べ高効率な分散処理形式が構成でき、各処理速度、実行速度が数段異なってくることです。また、68000 周波数スピードの選択や 6800 I/O 関係の選択は、最小システムと同様に選択して使用します。

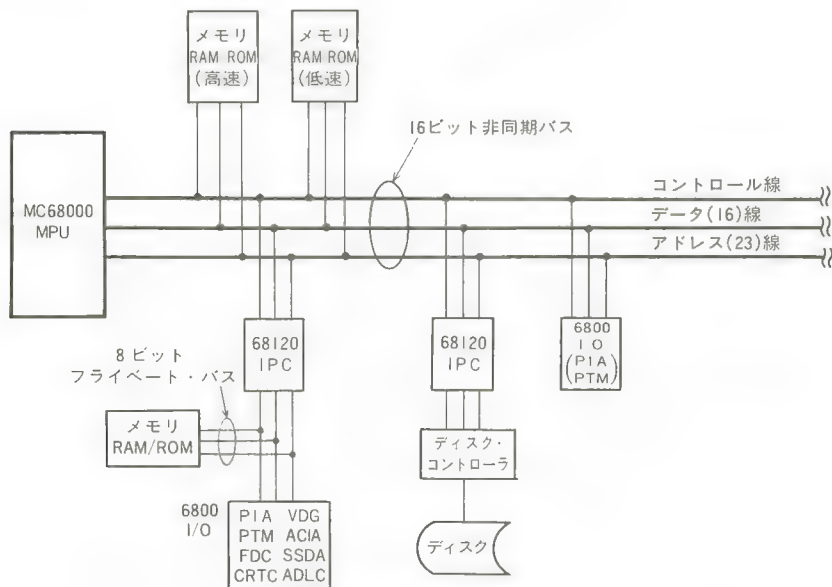


図 8・4 68000 中位システム

8.3 上位システム構成

上位システム構成は、マルチチャネルを備えることによってマルチユーザ・システムを構成し、リアルタイム・バス・ステートやリモート開発ステーションを備えることができます。

図8・5に示すように、中位システム機能のほかに、MC68450 DMAC, MC68451 MMU, MC68230 PITなどの機能が追加されます。メモリ管理ユニット(MMU)の追加によって、68000の16Mバイトのアドレス空間のアドレス変換とメモリ保護が強化され、しかも大容量のメモリを高効率に管理でき、オペレーティング・システムのオーバーヘッドを減少させることができます。また、ダイレクト・メモリ・アクセス・コントローラ(DMAC)の追加によって、四つの完全に独立したチャネルを持つことができ、4Mバイト/秒という高速なデータ転送が実行できます。

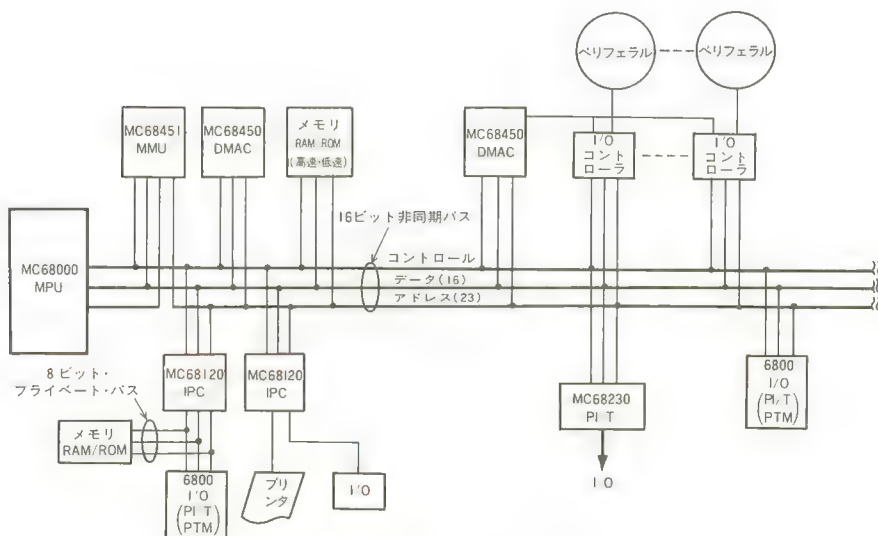


図 8・5 68000 上位システム

8.4 システムと VERSAbus の関係

VERSAbus は、スピード・アプリケーション用として、MC68000 用のボード、シャーシ、システム・ソフトウェアなどを応用機器製作要素としてモジュール化したファミリです。

バーサ・バスは、マルチプロセス・システムのバス・ストラクチャをとっており、8ビット、16ビット、32ビットの非同期式データと高速度のデータの転送を行います。そして、32アドレス・ラインによって、40億バイトのメモリを直接アクセスできます。またこれらのデータは、デバイス間のブロック・データ転送が5レベルのバスの優先スケマティックによってスムーズに行われます。そして、VERSAbus ストラクチャにおけるバス・クリア・コマンドを介してダイナミックに優先度を適応させます。

図8・6は、各種のモジュールとパフォーマンスの関係および推移を示したものです。また、価格とパフォーマンスの関係も選択の重要な要素の一つといえます。

図8・8は、バーサ・バスをベースに基本システムを構成した場合、拡張サポートの場合、マルチユーザ・システムの場合のブロック図とおおのこの関係を図示したものです。また、システムの拡張は単純であり、EXORmacs のシャーシに、

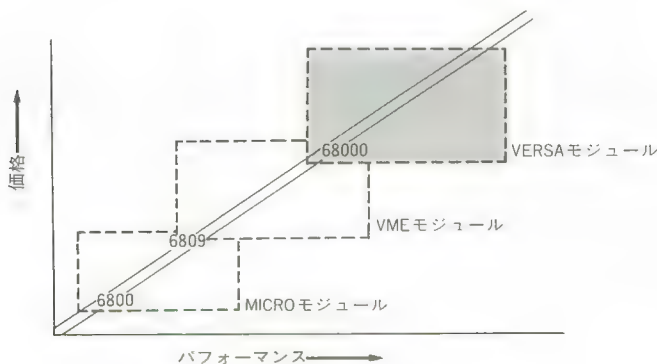


図 8・6 モジュールとパフォーマンスの関係

8 システムの構成

それぞれ適応するプラグ式のペリフェラルやモジュールを差し込むだけでよく、VERSdos オペレーティング・システムのアップデート・デバイス・テーブルと SYSGEN ユーティリティを用いて実行できます。

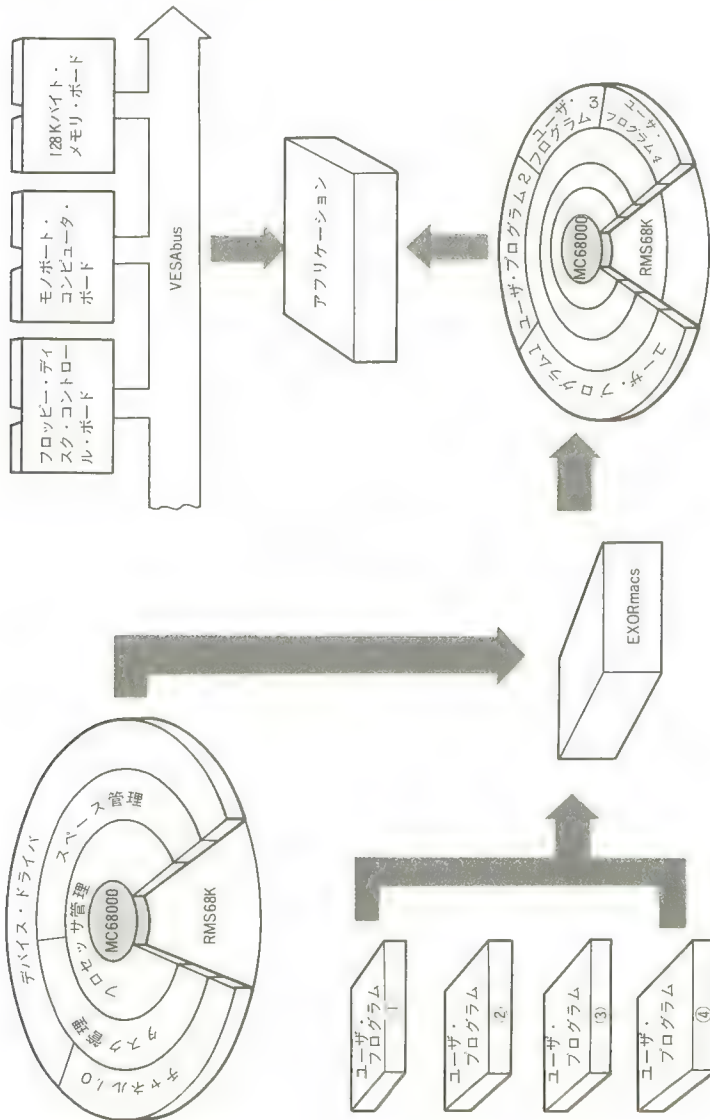


図 8.7 スピード・アプリケーション・パーサ・バス

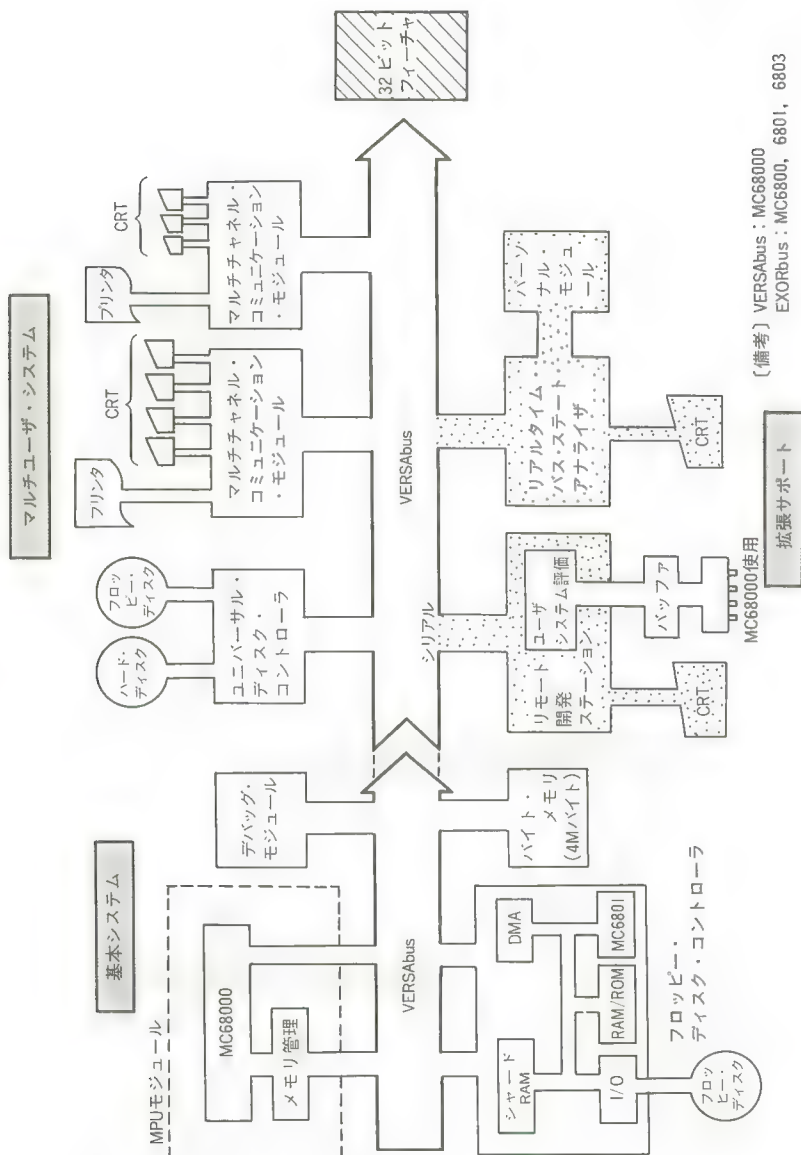
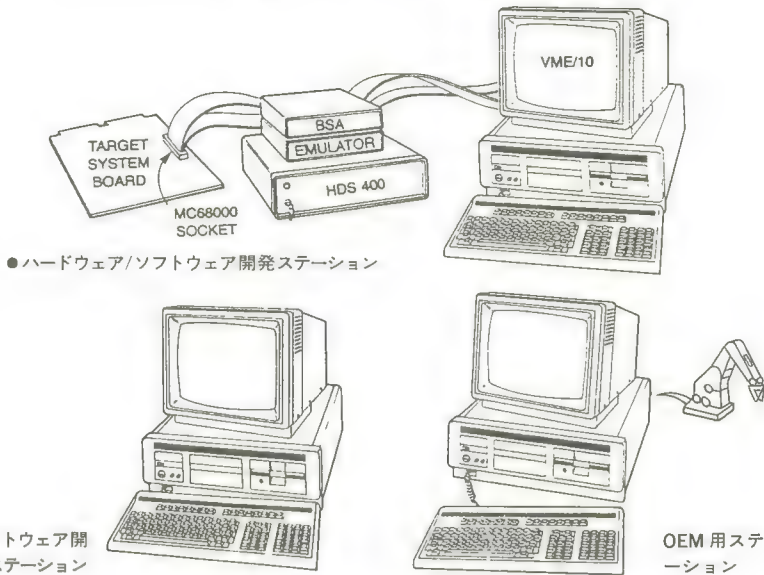
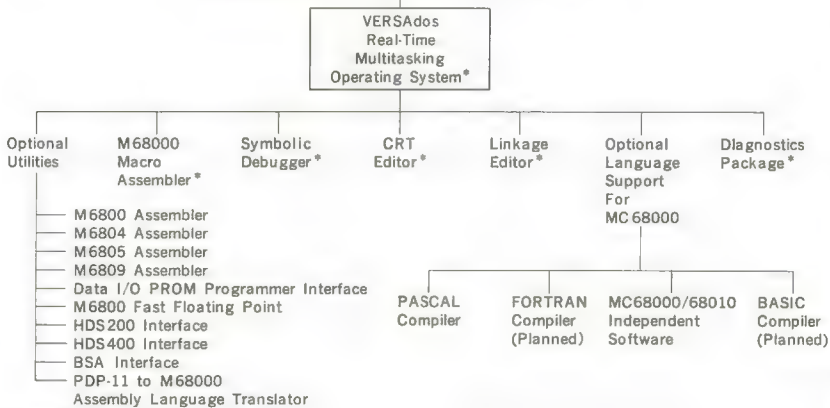


図 8・8 システムとの関係

VME/10 マイクロコンピュータ・システムの使用形態



VERSAdos オペレーティング・システムの環境

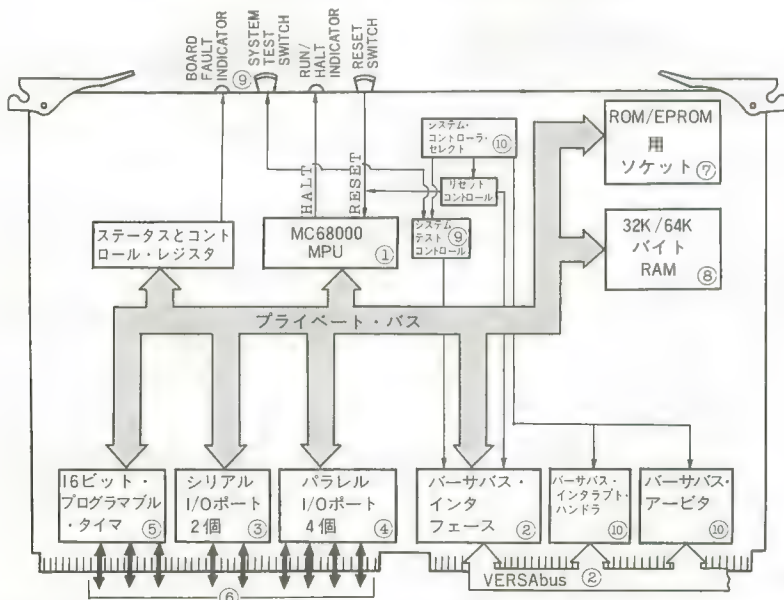


*標準で装備されています

VME/10 マイクロコンピュータ・システム (その3)

8 システムの構成

- ① MC68000 MPU (8 MHz)
 - ② VERSAbusインタフェース
 - ③ 2個のシリアルI/Oポート
 - RS-422用の同期式、非同期式のプログラマブルとストラバブル
 - ④ 4個のバラレルI/Oポート
 - おのおの8個のデータと2個のハンドシェーク・ライン
 - ⑤ 3×16ビットプログラマブル・タイマ
 - ⑥ バックプレーン 入力/出力 コネクタ
 - ⑦ ROM, EPROMソケット (64KMAX, 2, 4, 8Kバイト・ピンコンパチ)
 - ⑧ DRAM (32~64Kバイト)
 - ⑨ システム・テストとリセットのスイッチ、ボード・ステータス表示
 - ⑩ システム・コントロール・ファンクション
 - VERSAbusアービタ
 - VERSAbusインタラプト・ハンドラ
 - VERSAbusシステム・クロック、リセット、テスト、その他



8.9

9. 周辺ファミリ・チップ

いままで述べたように MC68000 は、他のプロセッサに類のない卓越した性能を持つマイクロプロセッサです。しかし、その性能をシステムで生かすには、周辺の LSI のサポートがあってこそです。この章では、これらの周辺チップの開発予定を明らかにするとともに、特に重要と思われる周辺チップについては、若干の機能の説明をしました。

9.1 MC68000をサポートする周辺チップ

MC68000を中心にしたマイクロコンピュータ・システムにおいて、システムの性能や信頼性向上は、MC68000 周辺チップのサポートを欠かすことができません。

モトローラ社は、図9.1に示すように、これらの周辺チップの開発、発売を予定しており、現在すでにMC68451 (MMU：メモリ管理ユニット)、MC68230 (PI/T：並列インタフェース用 IC タイマ内蔵)、MC68120 (IPC：インテリジェント・ペリフェラル・コントローラ) および数種の通信用 LSI を入手可能にしています。また、バーチャル機構を実現し、演算命令などの改良により実行スピードが向上した16/32ビット・マイクロプロセッサMC68010もすでに発売中です。MC68010は、MC68000とピン・コンパチブルで、命令などもアップワード・コンパチブルですので、MC68000で組まれたシステムをハードウェアの変更なしに置き換えるだけで、システムのパフォーマンス・アップを図ることができます。

MC68000 周辺チップは、大きく分けて三つに分類できます。それは、直接 MC68000 の出力する論理アドレスをシステムのグローバルの物理アドレスに変換するメモリ管理ユニットと、各種コントローラおよび各種インタフェース用チップです。

さらにMC68000ファミリは、単なる16/32ビットの新たなファミリ群として登場したわけではなく、8ビットのシステム・バスを持つMC6800ファミリに対して上位互換性を保っています。したがって、MC68000は、その同期式のバス・コントロール・ラインを用いて直接MC6800ファミリを接続し、周辺チップとして働かせることができます (図9.2参照)。

システムの性能向上の手段としては、クロック・スピードの速いMPUを用いることがあげられます。現状では、12MHzのクロック・スピードを持つMC68000 L12まで入手可能ですが、アクセス・タイムの遅いメモリや遅いクロック動作の周辺チップでは、スループットの向上は見込めません。周辺チップのクロック・スピードも向上する方向にありますが、今は8MHzが主流となっており、コス

9 周辺ファミリ・チップ

ト・パフォーマンスの点からも、システム・クロックを決める場合、周辺チップのクロック・スピードがかぎになっているのが現状です。

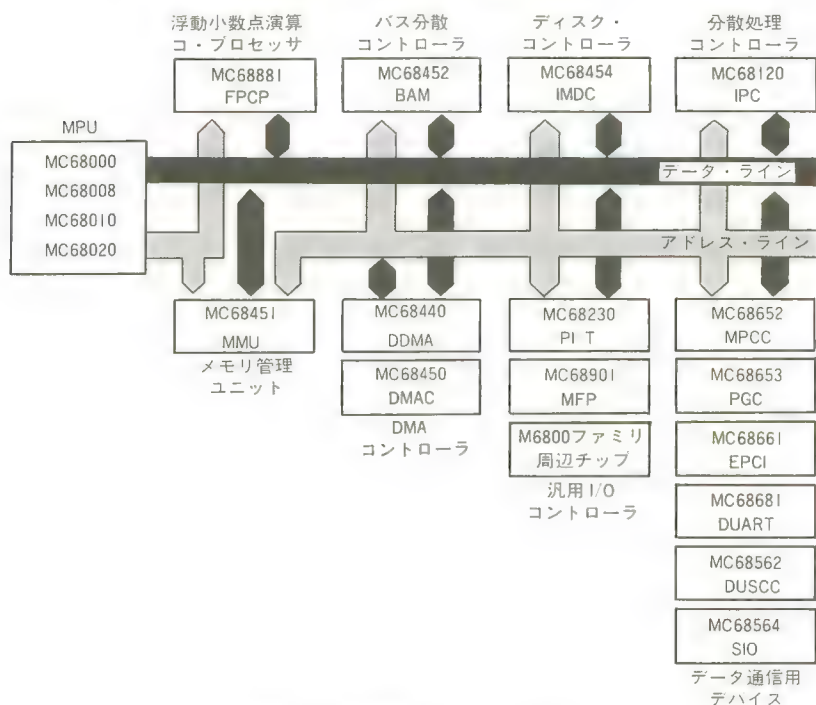


図 9-1 68000 ファミリ・チップ

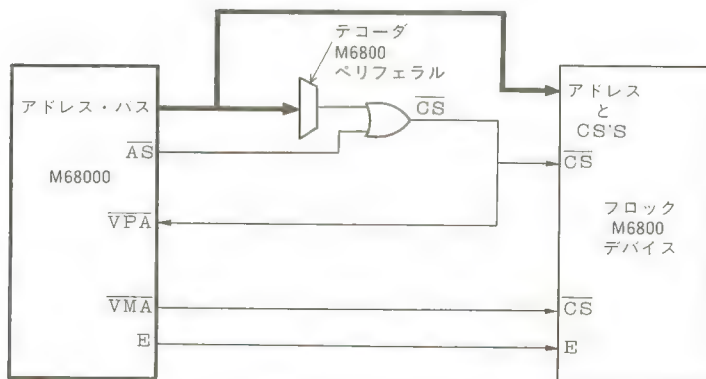


図 9-2 68000 と 6800 の接続ペリフェラル

9.2 メモリ管理ユニット MC68451

MC68451 は、MC68000 が出力する 16M バイトの論理アドレス空間を、実際に存在する物理アドレスに変換し、その物理アドレスの管理をする MC68000 周辺 LSI です。MC68000 が保有する 16M バイトのアドレス空間や、ユーザ・モード/スーパーバイザ・モードの二つの特権レベルの分離、そして強力なアドレッシング・モードなどの高性能な機能は、シングルユーザ/シングルタスク、バッチ指向の処理にとどまらず、ミニコン・クラス以上のコンピュータが実現していたマル

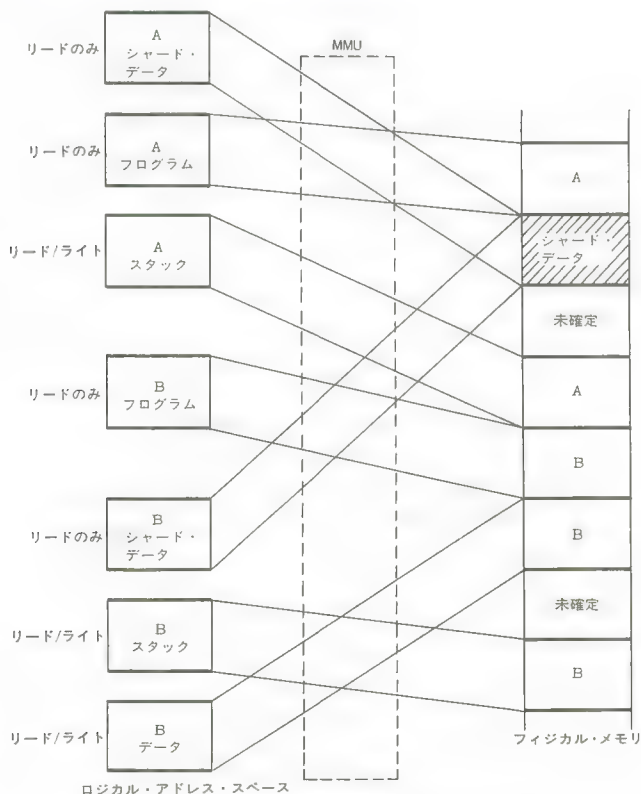


図 9.3 フィジカル・メモリにおけるマッピング・ロジカル・セグメント

チューザ/マルチタスク、リアルタイムなどの処理を十分サポートできるものです。これらのマルチ処理を行うには、当然、タスク間/システム・プログラムとユーザ・プログラム間/ユーザ間の相互アクセスをプログラマブルで禁止し、メモリ内の割付け、メモリの管理、保護を実行するメモリ管理用のチップMC68451が必要になります。

MC68451MMU (MEMORY MANAGMENT UNIT) の主な機能は、図9・3に示すような論理アドレスの、実際に存在する物理アドレスへの変換にあります。また、MC68000と1個のMMU、そしてメモリからなる最小システムを図9・4に示し、MMU 内部の機能ブロック図を図9・5に示します。

MMU は、メモリ空間がページ (256 バイト～16M バイト、 2^n バイト $8 \leq n \leq 24$) に分割された空間を、それぞれ論理/物理アドレス空間に割り付け、変換テーブルにより相互の対応をとるページング方式を用いています。

MMU は、図9・3に示す各アドレス空間の変換に際して、前もって内部のレジスタに設定された書込み保護、ファンクション・コードの指定 (スーパーバイザ/ユーザの分離、プログラム/データの分離)、アドレス空間の位置、大きさなどを

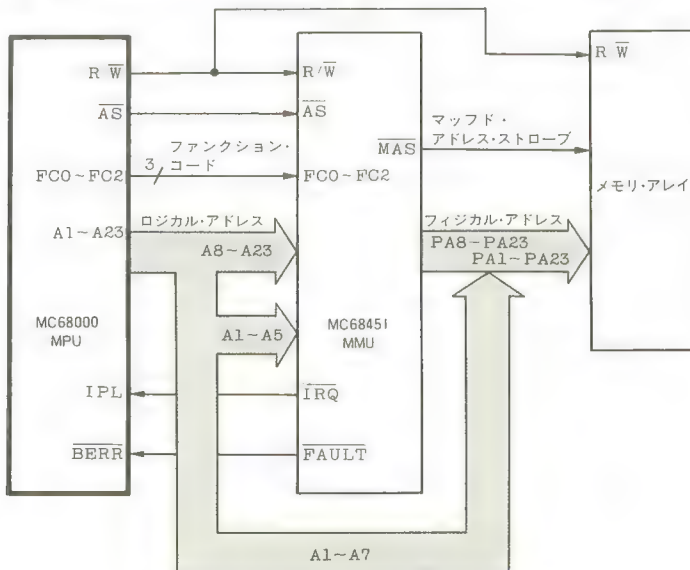


図 9・4 シングル MMU システム・ブロック図

9 周辺ファミリ・チップ

図9・5で示されるようなフローでチェックし、もしそれらに違反があると、割込みやフォルトなどでプロセッサに知らせます。

また、MC68451の特長として、一つのシステムに複数のMPUを共有できることがあげられます。おのののMMUは、アドレス空間を32個のセグメントに分割して管理、保護する能力を持っていますが、それ以上のセグメントが必要な場合に複数のMMUを使用します。

MC68451は、前に述べたセグメント・フォルト（書込み/セグメント範囲違反など）をプロセッサに知らせる機能により、仮想記憶制御に対応します。しかし、それにはプロセッサ自体にセグメント・フォルトの要因解除後の再開始機能が含

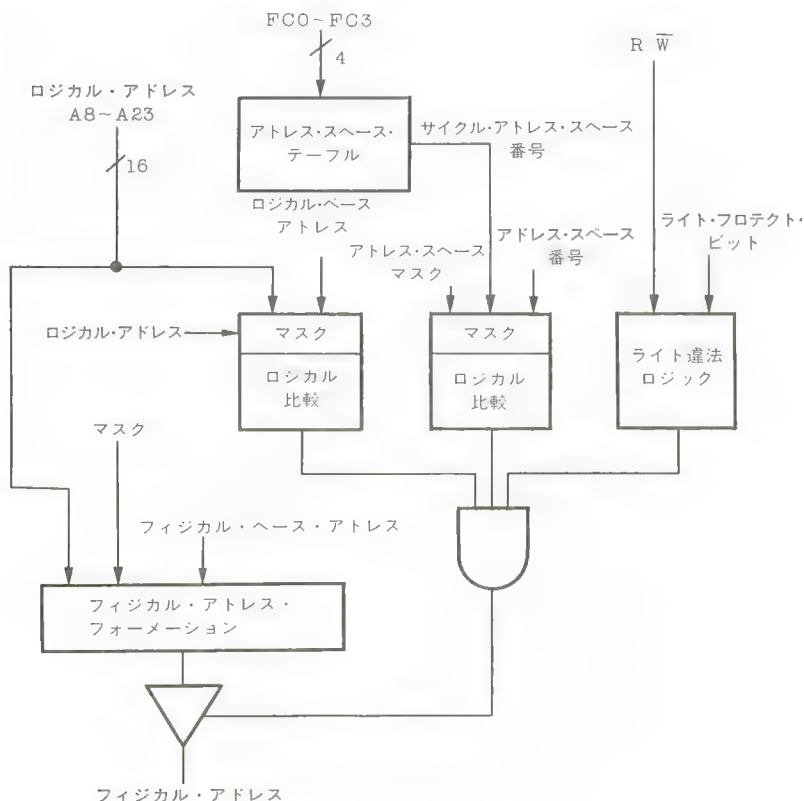


図 9・5 MMUのファンクション・ブロック図

9 周辺ファミリ・チップ

まれていなければなりません。この機能を実現するには、MC68010（バーチャル・マシン）が持つ例外処理の強化や、フォルト発生時、プロセッサが内部状態（データ/アドレス・レジスタ、プログラム・カウンタ、ファンクション・コード、中断された命令など）を再開可能に十分なだけスタックする機能を備えることによりなされます。

MC68000 は、MC68010 のような命令の再開機能は備えていませんが、図 9・6 のように 2 台のプロセッサを用いて仮想記憶制御を実現します。図中の MPU1 が通常の処理を行う主プロセッサとなり、MPU2 が、MPU1 をバック・アップして仮想記憶制御を実行します。MPU2 は、セグメント不在のフォルトをバス・エラー入力により検出すると、MPU1 への \overline{DTACK} （データ転送アクノリッジ）入力信号を強制的に返さないようにし、MPU1 をウエート・ステート（ \overline{DTACK} 待ち）状態にします。この間 MPU2 がバス・マスタとなり、二次記憶機器などのハード・ディスクから主記憶上に不在であったデータを主記憶にセグメント・スワップし、MMU のセグメント・ディスクリプタ更新をします。その後 \overline{DTACK} を MPU1 に返すことにより、MPU1 は中断されていた命令を開始し、通常の処理を続行します。

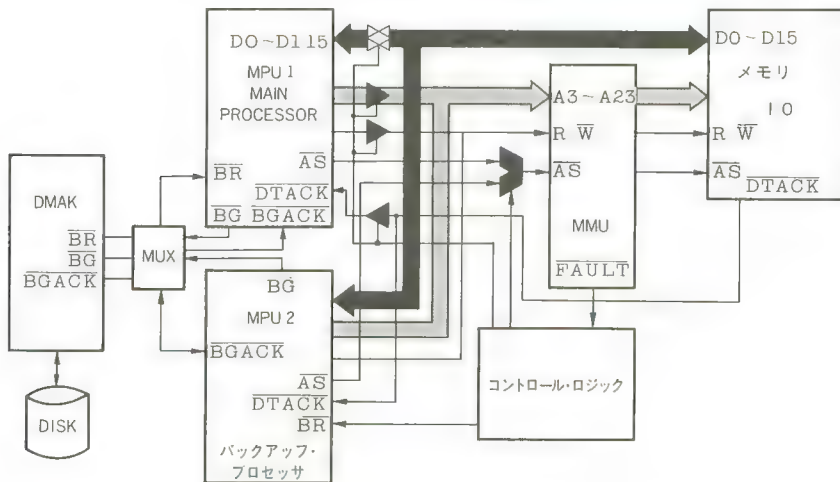


図 9・6 MC68000 と MC68451 におけるバーチャル・メモリ例

9 周辺ファミリ・チップ

9.3 DMAコントローラ MC68450

16ビットのマイクロコンピュータ・システム，特に MC68000 を CPU とした高性能なミニコン・クラスのシステムでは，大量のデータが必要となります．また，リアル・タイム処理を実行するにあたっては，入出力機器から随時データを入力し，システムの主記憶に高速データ転送をしなければなりません．

このように高速 CRT 端末，ハード・ディスクおよびフロッピー・ディスクなどの大量のデータ入出力機器を持ったシステムや，メモリ内をタスク分けしてデータやプログラム域の整備を必要とするシステムでは，入出力機器と主記憶，あるいは主記憶間での大量のデータ転送が要求されます．MC68000 自身も，MOVE

表 9.1 DMAコントローラの比較

項 目	単 位	MC68450	i8089	MC6844
クロック周波数	MHz	4, 6, 8 (予定)	5	1, 1.5, 2
パッケージ・ピン数	本	64	40	40
チャンネル数	本	4	2	4
直接アドレス空間	バイト	16M	1M	64K
転送単位	ビット	8/16/32	8/16	8
転送語数	語	64K	64K	64K
最高転送速度	M バイト/秒	4 (8MHz 版)	1.25	2 (2MHz 版)
対応デバイス	ビット	8/16	8/16	8
複数ブロック転送機能		アレイ・チェーン，リンク・アレイ・チェーン，コンティニュー	ソフトウェア・プログラマブル	コンティニュー
その他		<ul style="list-style-type: none"> ▷ リトライ機能，バース・エラー機能などの例外処理機能 ▷ 動作タイミング・エラー，カウント・エラー，コンフィギュレーション・エラーなどのエラー検出機能 ▷ アドレス/データ多重 	<ul style="list-style-type: none"> ▷ I/O プロセッサとして動作する ▷ トランスレート機能 ▷ サーチ機能 ▷ I/O アドレス空間とメモリ・アドレス空間を分離 ▷ アドレス/データ多重 	<ul style="list-style-type: none"> ▷ メモリ-メモリ間転送機能なし ▷ HMCS6800 周辺 LSI，ただし HD 68000 と直接インタフェース可能

9 周辺ファミリ・チップ

命令などの転送命令により、データ転送を実行することができますが、アドレス情報、転送語長および転送後の語長計算をして転送終了をソフトウェアですべて考慮しなければならないため、大量の高速データ転送は見込めません。

そこで、マイクロプロセッサに代わってバス占有権を奪い、この高速データ転送を制御するMC68000 周辺 LSI であるDMAC (DIRECT MEMORY ACCESS CONTROLLER) が登場します。いままでに、8ビット・マイクロプロセッサ用 DMA コントローラやインテルi8089 I/O コントローラなどが製品化されていますが、MC68450 は、16ビットのDMA 転送専用コントローラとしては初めてのもので、入出力機器制御用のプロセッサとして働き、その手段としてDMA 転送機能を備えているi8089 とは、基本的に異なったLSI です。

各種DMA コントローラの機能比較を表9・1 に示します。

MC68450 は、MC68000 の持つ高信頼性システム設計を反映しており、他の

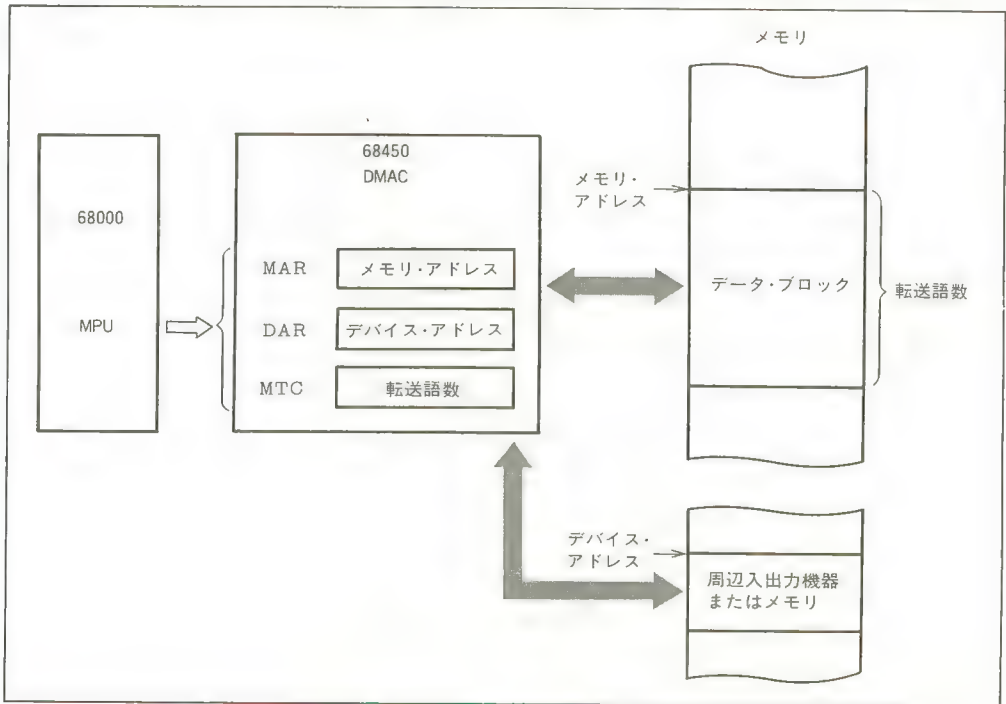


図 9・7 単一ブロック転送

9 周辺ファミリ・チップ

DMA コントローラにはない例外処理，エラー検出機能や，外部アボート入力やソフトウェアによるホルト，アボートなどの機能を備え，しかも最高転送速度 4M バイト/秒を実現し，従来の DMA コントローラの性能を大幅に超えたものといえます．MC68450 は，1 チップに独立したチャンネルを四つ内蔵し，各チャンネルの優先順位もソフトで設定できます．

転送データの単位は 8，16，32 ビットで，32 ビットのデータ転送では，二重アドレッシング・モードを用いて実行します．

MC68450 は，四つの完全に独立したチャンネルをそれぞれ異なる入出力機器を接続し，連続したデータ群（データ・ブロック）の転送を，以下に述べる 3 種類のモードで実行します．

（1）コンティニュー・モード（図 9・7 参照）：単一ブロック転送を MPU の指示により自動的に繰り返す転送方式．単一ブロック転送とは，メモリおよびデ

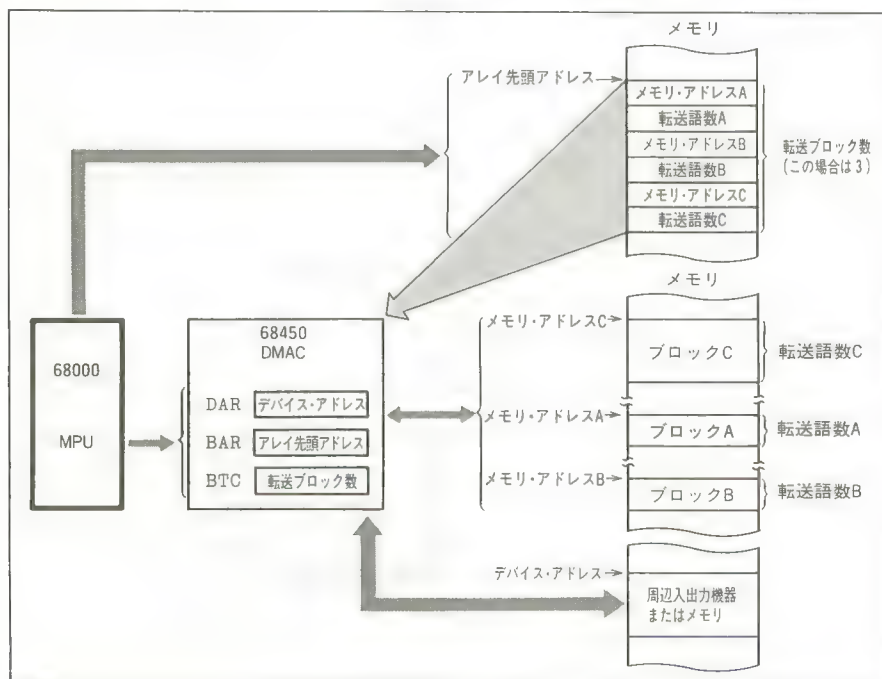


図 9・8 アレイ・チェーン転送

バイスの転送元あるいは転送先の先頭アドレスと転送語数をDMA（内の各レジスタにMPUが設定し、1データ・ブロック転送する方式。

（2）アレイ・チェーン・モード（図9・8参照）：MPUがあらかじめメモリ内にブロック情報をアレイ状に並べたテーブルを作成し、DMACの初期化（レジスタ設定）をすることにより、DMACが各ブロック転送ごとにテーブルを参照し、複数のデータ・ブロック転送を自動的に行う方式。

（3）リンク・アレイ・チェーン・モード（図9・9参照）：アレイ・チェーンと同様に複数のデータ・ブロック転送に使用されます。両方式ともに各ブロックの先頭アドレスと転送語数をメモリ内に収納していますが、リンク・アレイ・チェーン・モードでは、さらにリンク・アドレス（次のブロックの転送情報の入っているアドレス）も収納しています。したがって、この場合、アレイ・テーブルが連続する必要がなく、ブロック転送の削除や挿入が、リンク・アドレスを書き換え

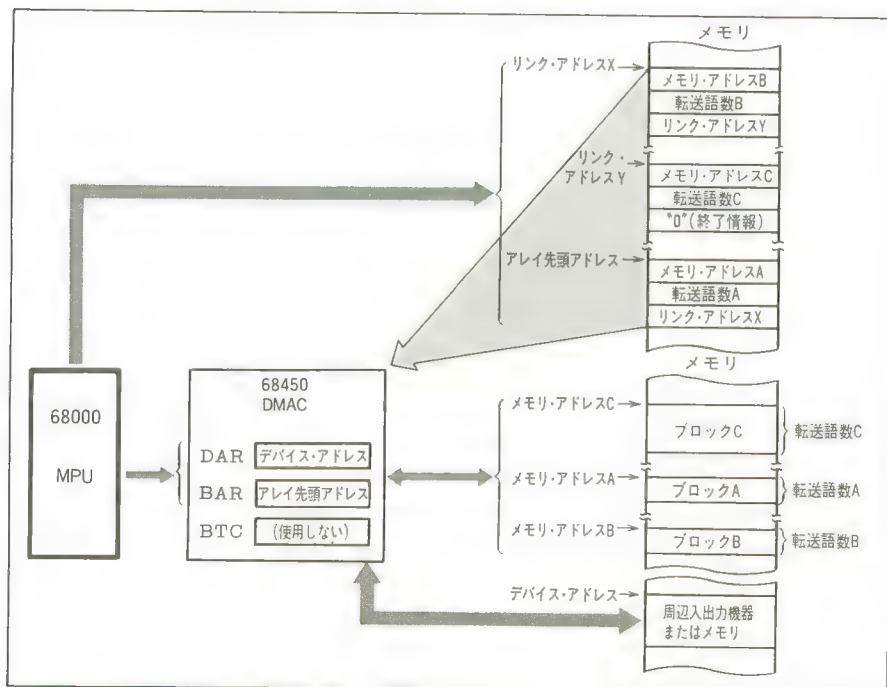


図 9・9 リンク・アレイ・チェーン転送

9 周辺ファミリ・チップ

るだけで簡単にできます。

また、DMAC がデータ転送要求を受け付ける方法も、大きく分けて以下の三つがあります。

(1) バースト・モード：連続するバス・サイクルを占有し、連続してデータ転送する方式。

(2) サイクル・スチール・モード：1 データ単位ごとに DMAC が MPU からバス占有権を奪いデータ転送をする方式。

(3) オート・リクエスト・モード：DMAC が内部的に転送要求を出す方式。

9.4 MC68000 周辺通信用 LSI

現在、コンピュータ・システムが扱う情報は、増大/高度化/多様化の方向にあります。そのシステム間の正確な高速データ通信は、情報の停止（腐沈化）を防ぐとともに資源共用化を図ることができ、コンピュータ・ネットワークの信頼性や性能を決める重要な役割を果たします。

データ通信方式には、大きく分けて非同期式（調歩同期）通信方式と同期式通信方式の二つがあります。また、同期方式の通信においても、同期方式やデータのフォーマッタの違により、HDLC、SDLC、BISYNC、イーサネットなど各種プロトコルの通信があります。

MC68000 ファミリ・データ通信用 LSI は、これらの通信方式の一つ以上を満足するように設計されています（図 9・10 参照）。

また、各データ通信用 LSI の機能比較表を表 9・2 および表 9・3 に示しました。

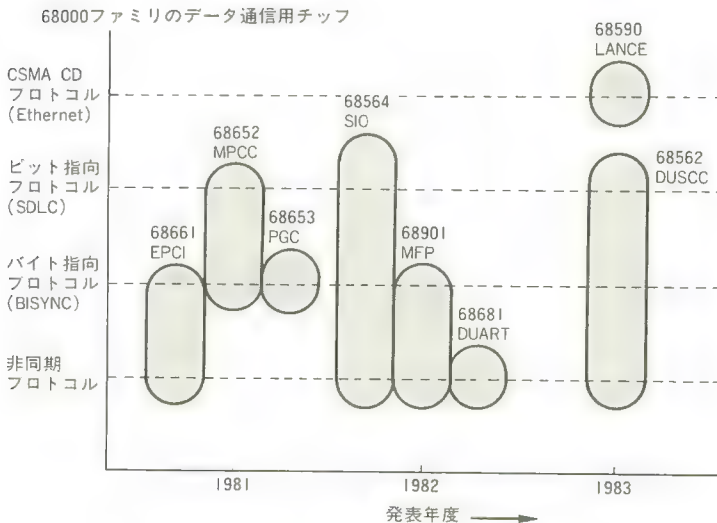


図 9・10 68000 ファミリのデータ通信用チップの方向

9 周辺ファミリ・チップ

表 9・2 68000 データ・コミュニケーション・コンバージョン

	68661 EPCI	68681 DUART	68652 MPCC	68564 SIO	68562 DUSCC
68000 INTERFACE	NO	YES	NO	YES	YES
ASYNCHRONOUS	YES	YES	NO	YES	YES
STOP BITS					
1,1,5,2	YES	YES	NO	YES	YES
INCREMENTS	NO	YES	NO	NO	YES
PARITY	YES	YES	NO	YES	YES
SYNCHRONOUS	YES	NO	YES	YES	YES
BOP	NO	NO	YES	YES	YES
BCP	YES	NO	YES	YES	YES
CHANNELS	1	2	1	2	2
MAX RATE(MBITS/S)	1	1	2	1	4
BUFFERING R/T	2/2	4/2	1/1	4/2	4/4
DATA BUS	8	8	8/16	8	8
PROG BAUD RATES	YES	YES	NO	NO	YES
CRC	NO	NO	YES	YES	YES
INTERRUPTS V & AV	NO	V	NO	V & A	V & A
INTERRUPT REGS	0	4	0	2	5
REGISTERS	12	17	4	21	27
PROG TIMER	NO	YES	NO	NO	YES
CHAR LENGTH	5-8	5-8	1-8	5-8	5-8
PROTOCOLS	S=SUPPORTED ON CHIP		C=CAPABLE OF PERFORMING		
BOP					
ACCCP			C	C	C
HDLC			C	S	C
SDLC			C	S	S
X.25			C	C	C
BCP					
BISYNC	C		C	C	S
DDCMP	C		C	C	C
PINS	28	40	40	48	40/48

表 9・3 68000 データ・コミュニケーション・サポート

SPEED	PROTOCOL	DEVICE
0 TO 1M BPS	ASYNCHRONOUS	MC6850
		MC68562
		MC68564
		MC68661
		MC68681
	BCP-BISYNC	MC6852
		MC68562
		MC68564
		MC68652
		MC68661
	BOP-SDLC/HDLC/X.25	MC6854
		MC68562
		MC68564
		MC68652
1M TO 2M BPS	BCP	MC6852(4.5M)
		MC68562
		MC68652
	BOP	MC6854
		MC68562
		MC68652
2.0 TO 4.0 BPS	BCP BOP	MC68562
		MC68562

9.5 MC68000 と MC6846 のインタフェース例

MC6846 ROM I/O タイマ (RIOT) と 68000 のインタフェース例を図 9.12 に示します。MC6846 RIOT は、2K×8 のマスク・プログラム ROM と 8 ビットの I/O ポート、16 ビットのプログラマブル・タイマ/カウンタを内蔵した 40 ピンのパッケージです。

図 9.12 は、68000 と RIOT とのインタフェースを、下位の 10 本のアドレス・ライン (A1～A10) と 16 本のデータ・ライン (D0～D15)、リード・ライト線 (R/W)、リセット (RESET)、イネーブル (E)、チップ・セレクト信号を用いた TVBug 回路です。

図 9.13、図 9.14 に示すように、RIOT は同期式、非同期式でも 68000 とインタフェースが行えます。ROM セレクト用と I/O タイマ用として CS0 上位側と CS1 下位側があり、アドレスは、上位側の A6～A10、下位側の A3～A5 と、I/O コントロール・レジスタ用の A0～A3 があります。

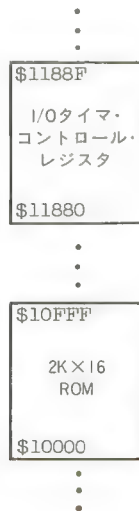


図 9.11 メモリ・マップ

9 周辺ファミリ・チップ

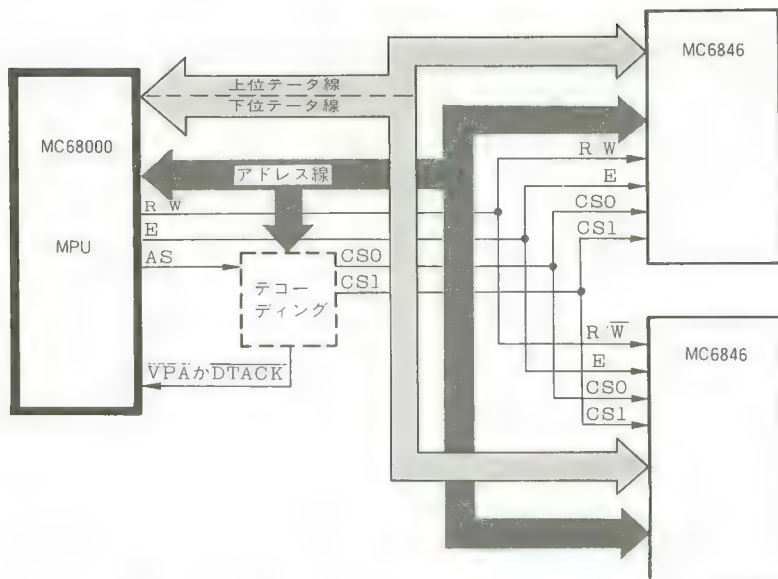


図 9・12 68000とMC6846のインタフェース・ブロック図

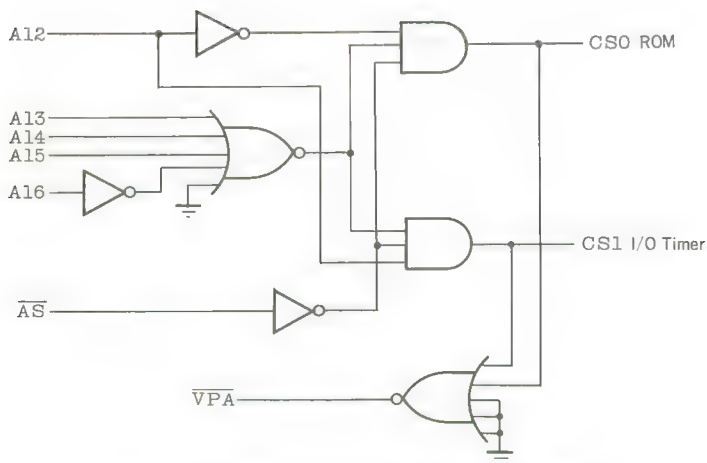


図 9・13 同期式インタフェースのデコーディング回路

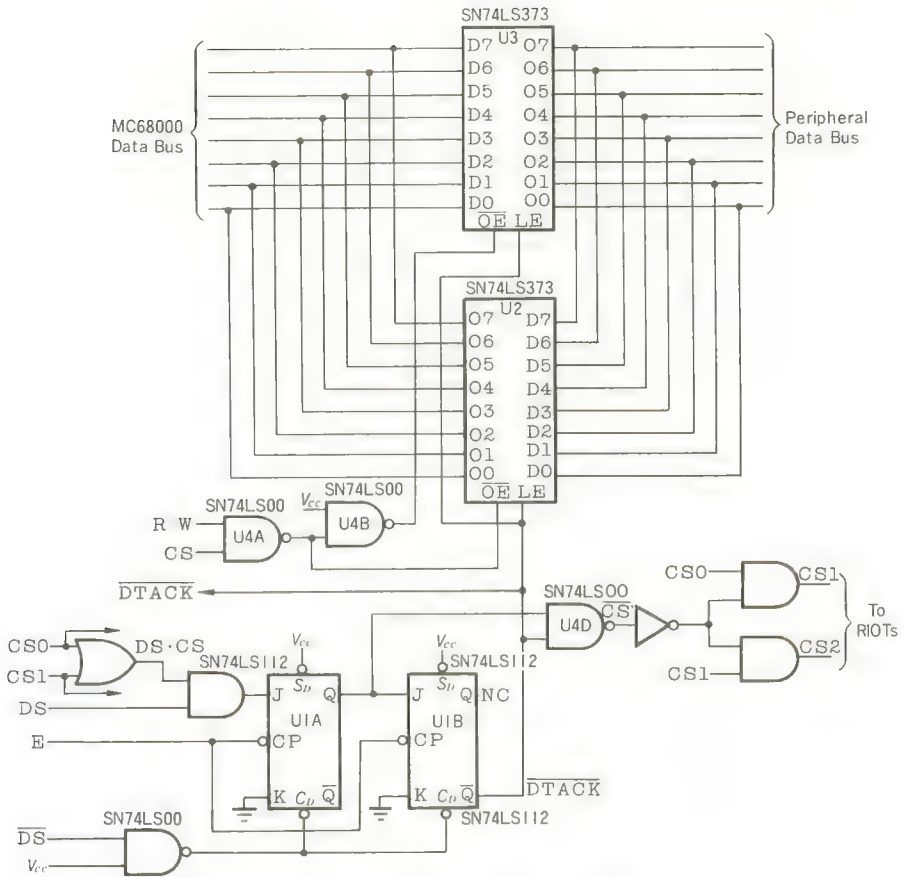


図 9・14 非同期式インタフェース回路

9 周辺ファミリ・チップ

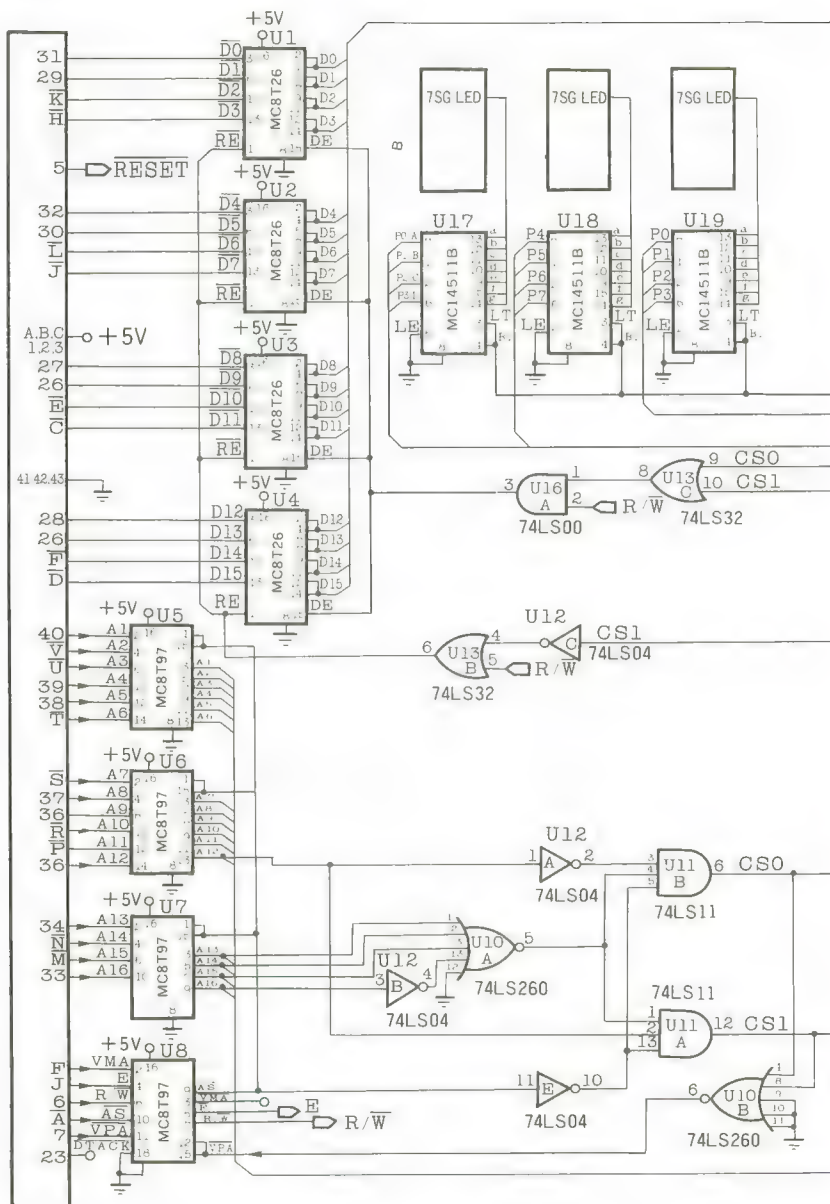
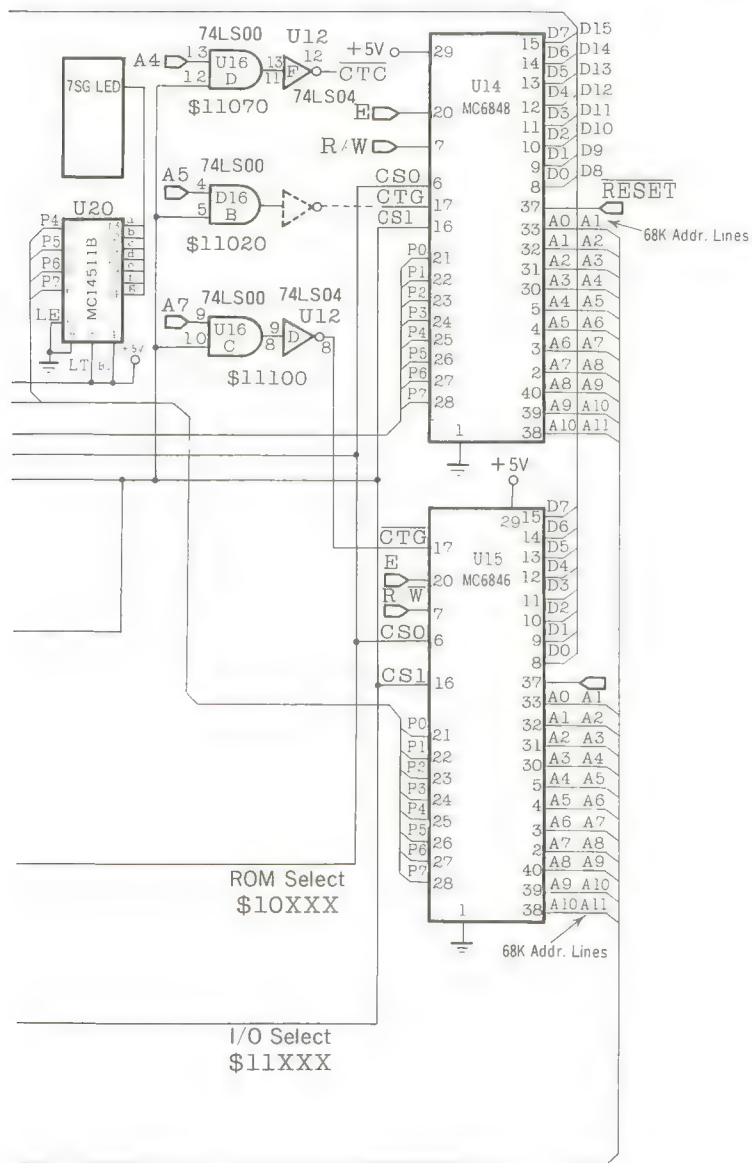


図 9・15 68000 と MC 6846



インタフェースのスキマティック


```

37 001042 02430F00      AND.W      #$0F00,D3
38 001046 0C430900      CMP.W      #$0900,D3
39 00104A 6F000006      BLE         BYTE4
40 00104E 04430800      SUB.W      #$0800,D3
41 001052 8443          OR.W       D3,D2
42 001054 2601          MOVE.L    D1,D3
43 001056 02830000F000  AND.L      #$F000,D3
44 00105C 0C8300009000  CMP.L      #$9000,D3
45 001062 6F000006      BLE         DONE
46 001066 04438000      SUB.W      #$8000,D3
47 00106A 8443          OR.W       D3,D2
48 00106C 33C200011886  MOVE.W     D2,PDR
49 001072 2E3C0003D090  MOVE.L     #250000,D7
50 001078 5387          SUB.L     #1,D7
51 00107A 6AFC          BPL        DLY
52 00107C 51C8FF9C      DBRA       D0,BYTE1
53
54
55      *      BACK TO MACSBUG
56
57 001080 4E4F          TRAP       15
58 001082 0000          DC.W      0
59
60      *      THE PIA ADDRESSES ARE
61      *      $11882
62      *      $11884
63      *      $11886
64      *      TO WRITE TO THE DISPL
65
66      *      END
67
68 ***** TOTAL ERRORS 0-- 0

```

USE LONG WD. FOR MSB.

DISPYD DATA IN D2
WRITE DATA TO PIA
DLY FOR 5 SEC

GO AGN IF <> 10

図 9.16 サンプル回路における 68000 プログラム例

SYMBOL TABLE

BYTE1	00101A BYTE2	00102C BYTE3	00103E BYTE4	001052
CVRCHK	001014 DDR	011884 DLY	001078 DONE	00106A
PCR	011882 PDR	011886 ROMSTR	010000	

MC 68010 仮想マシン

MC 68010 は MC 68000/68008 の機能と特徴を完全にカバーし、Virtual Memory/Machine をサポートします。

68000 と比べての主な追加点と変更点

(1) 二つのレジスタの追加

VBR: ベクタ・ベース・レジスタ

(ベクタ・テーブルをリロケータブルにする)

SFC, DFC: オルタネイト・ファンクション・コードレジスタ

(オルタネイト・アドレス・スペースへの転送)

(2) 新規インストラクションの追加

RTD: Return and Deallocate Stack

MOVES: Move Alternate Address Space

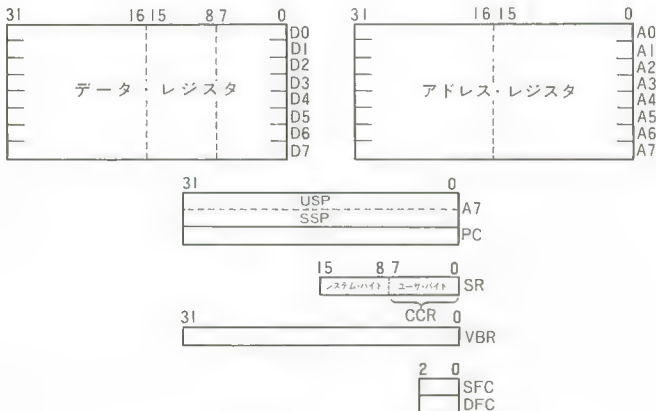
MOVEC: Move Control Register

MOVE from CCR: Move from Condition Code Register

(3) 1 部のインストラクションがスピード・アップされた。

全体的に 20~25% のスピードアップ

(4) 特権命令 MOVE from SR 命令が特権命令に変更



10. MC 68000 開発装置

68000 の開発支援装置であるエクサマクス (EXORmacs) とボックス・コンピュータ (VMC 68/2) の構成と機能について、それぞれやさしく図示し説明します。そして、エクサマクスとボックス・コンピュータの特長と用途も説明します。また、用途、方法をより理解していただくために、EXORmacs の実際の使用例、ディスク・オペレーティング・システム (VERSA dos) も載せておきました。

10.1 エクサマクス

エクサマクス (EXORmacs) は、MC 68000 (16/32 ビット・プロセッサ) のソフトウェアとハードウェア設計者による開発や設計を容易にするためのサポート・システムです。ハードウェアのアーキテクチャは、今後のマイクロコンピュータにおいて要求されるハイ・パフォーマンスをサポートできます。また、マルチタスキング・オペレーティング・システムは、フロッピー・ディスク・ベース・システムおよびハード・ディスク・ベース・システムをサポートします。ハード・ディスク・ベースとフロッピー・ディスク・ベースとの相違は、供給されるドライブ・ユニットと“VERSAdos-E”オペレーティング・システムによって決定されます。

[1] エクサマクスの特長

(1) MPU に MC 68000 を搭載し、MC 68000 のための最大 8 ステーションのマルチユーザ開発システムです。

(2) マルチタスク・オペレーティング・システムとメモリ・マネージメント機能を持ち、I/O 分散処理方式を備えています。

(3) 構造化文が使えるマクロ・アセンブラと PASCAL 高級言語が使える、シンボリック・デバッグが可能です。

(4) ハード・ディスク・サブ・システムは、2 台のドライブをサポートし、記憶容量は 32 M バイトから 192 M バイトまで可能です。また、1 M バイトのフロッピー・ディスクをサポートし、2 M バイトまで拡張が可能です。

(5) おおの 16 M バイトのメモリ空間を持つ一次マップ、二次マップの二重メモリ・マップ機能をサポートし、工業規格のバーサ・バス (VERSAbus) によって、32 ビットのデータ・バスをサポートします。

(6) 自己診断ファームウェアによるセルフ・テストやフロント・パネルで動作状態やエラー箇所が表示されます。

(7) ホスト・コンピュータからオンライン・ダウン・ロードが可能です。

(8) 132 桁のプリンタの駆動ができ、スピードは、180 ch/600 line・s の 2 種

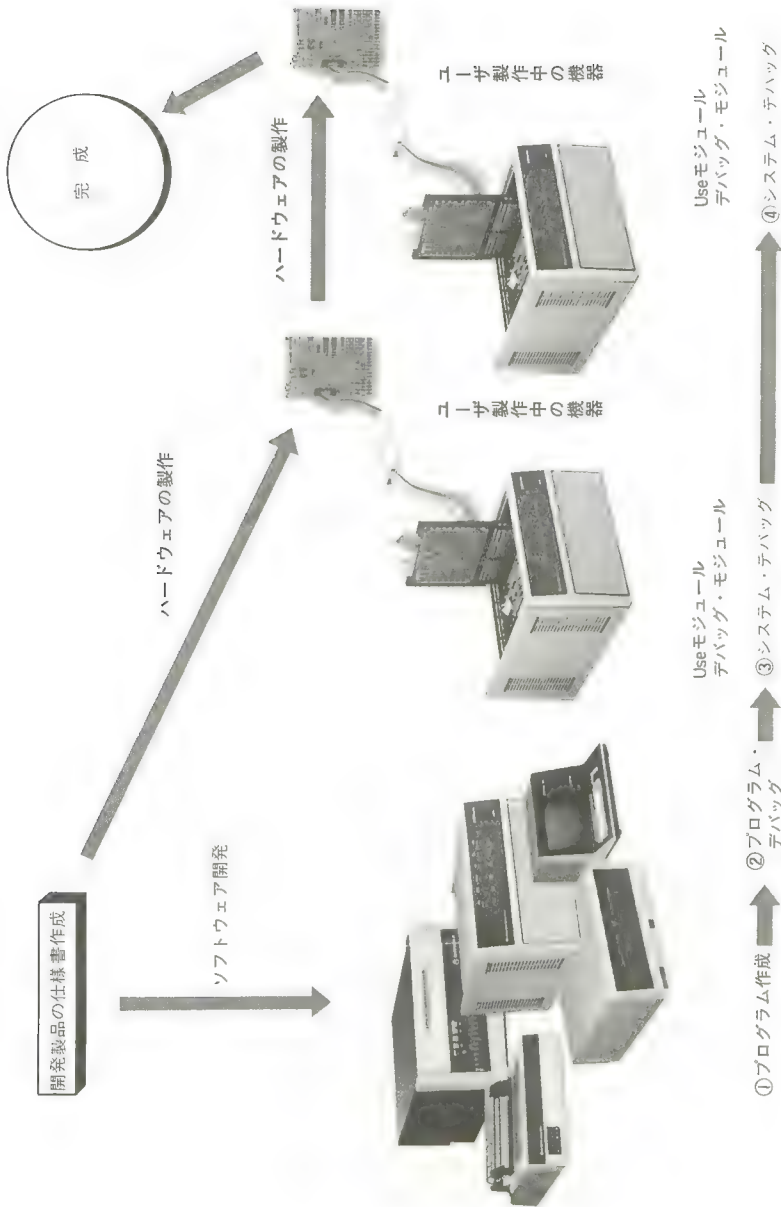


図 10・1 開発のフロー

IO MC68000 開発装置

類のプリンタをサポートし、インテリジェントCRTによってソフト・キー入力ができます。

エクサマックスのシステム構成は、表 10・1 に示すようにフロッピー・ディスク・

表 10・1 エクサマックスの構成

構成

	フロッピー・ディスク・タイプ	ハードディスク・タイプ	
タイプ	M68KMACSS3/M68KMACSF1	M68KMACSH1	M68KMACSH1A
CPU	EXORmacs 本体		
CRT	EXORterm 155		
DISK	EXORdisk III (1.0 Mbyte フロッピー・ディスク)	32Mbyteハード・ディスク (16 Mbyte 取外し可/ 16 Mbyte 固定)	96Mbyteハード・ディスク (16 Mbyte 取外し可/ 80 Mbyte 固定)
SOFTWARE	Structured Macro Assembler/Linkage Editor/ PASCAL Compiler/CRT Editor/Symbolic Debug/ VERSAdos Operating System		

M68KMACSS3 はプリンタ付きです。

※マルチ・ユーザ処理を行うには、EXORmacs 本体に M68KMCCM (Multi-Channel Communication モジュール) を増設することにより、1 モジュールで 4 ステーションと 1 プリンタが追加できます (最大 3 モジュールで 12 ステーションと 3 プリンタまで接続可能)。

※フロッピー・ディスクとハード・ディスクは 60Hz 用と 50Hz 用に区別されています。

表 10・2 エクサマックスのオプション

オプション・モジュール

M68KRDS	Remote Development Station
M68KUSE	MC 68000 User System Emulator
M68BSAC	Bus State Analyzer Control
M68BSA1	MC68000 Bus State Analyzer Personality
M68BSA2	MC6800/6809/6829 Bus State Analyzer Personality
M68BSA4	MC68120/6801 Bus State Analyzer Personality
M68BSA5	VERSAbus Bus State Analyzer Personality
M68KVM11-1	256Kbyte Dynamic RAM Module
M68KVM11-2	512Kbyte Dynamic RAM Module
M68KVM10-3	128Kbyte Dynamic RAM Module
M68KVM10-2	64Kbyte Dynamic RAM Module
M68KVM10-1	32Kbyte Dynamic RAM Module
M68KVAM	VERSAbus Adapter Module
M68KEXTM	VERSAbus Extender Module
M68KWW	VERSAbus Wirewrap Module
M68KMCCM	Multi-Channel Communication

タイプとハード・ディスク・タイプの2タイプがあります。また、オプション・モジュールを表10・2に示します。

〔2〕 EXORmacs の装置

(1) シャーシは、モジュール・ハウジング、電源、空冷ファン、フロント・パネルなど。

(2) デバッグ・モジュールは、EXORterm 155 ディスプレイ・コンソールと、ホスト・コンピュータ用の二つの RS-232 シリアル・ポートと、703, B600 プリント用のポートと、合計三つのポートを備えています。

(3) マイクロプロセッサ・モジュールは、4セグメントのメモリ管理ユニットと診断用ファームウェアを備え、マルチ・タスク・オペレーティングの保護をします。

(4) フロッピー・ディスク・コントローラ・モジュールは、4 台までのフロッピー・ディスク・ドライブのサポートをします。

(5) 128 Kバイト・ダイナミック・メモリ・モジュールは、オペレーティング・システムとサポート・ソフトウェアを格納します。

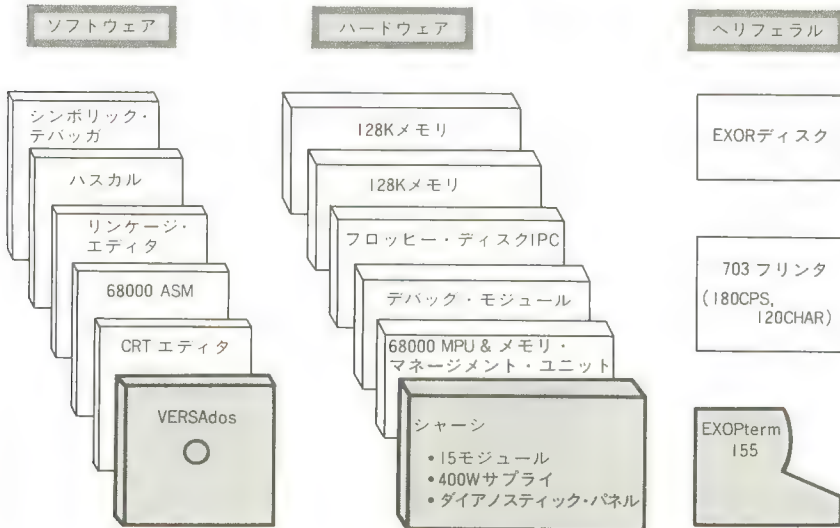


図 10・2 EXORmacs システムによるハードウェア、ソフトウェア開発装置

10 MC 68000 開発装置

(6) 汎用 IPC モジュールは、ユニバーサル・ディスク・コントローラ (UDC) の一つであり、ハード・ディスクとフロッピー・ディスクを制御するソフトウェアも含んでいます。

(7) ディスク・インタフェース・モジュールは、標準 DMA を介して汎用 IPC へのインタフェースを行います。

[3] EXORmacs のシステム

(1) オペレーション・モードは、二つのモードを持ち、オペレーティング・

表 10・3 16/32 bit 開発支援セレクション・ガイド

	シングル・ユーザ・システム (2ターミナルで使用可)	4ユーザ・システム	8ユーザ・システム
基本構成	M68KMACSS3 または M68KMACSF1	M68KMACSH1 M68KMCCM×1 M68SXD10155×4 M68KVM11-1×1 M68KFD1102	M68KMACSH1A M68KMCCM×2 M68SXD10155×8 M68KVM11-2×1 M68KFD1102
周 辺	M68K703LP1 M68SXD10155 M68KFD1102		
メモリ	M68KVM10-1 M68KVM11-1 M68KVM10-2 M68KVM11-2 M68KVM10-3		
マルチユーザ への拡張	M68KHDS32-1 または M68KHDS96-1 M68KMCCM		
マス・ストレ ージの拡張	M68KHDE32-1 M68KHDE96-1 M68KFD1102 M68SFDU1102E		
オプション・ ソフトウェア	FORTRAN MPL* COBOL* BASIC-M UNIX*	ADA 各種8ビットMPUアセンブラ APL* FORTH*	
オプションⅠ	M68KUSE M68KEXTM M68KWW M68KRDS1	M68KMACSRK M68KVAM	M68BSAC M68BSA1 M68BSA2 M68BSA4 M68BSA5
オプションⅡ	VERSAmdules, Micromodules ファミリを使用可		

* 他社から供給の予定

(注) フロッピー・ディスクとハード・ディスクは、50Hz と 60Hz 用に区別されております。

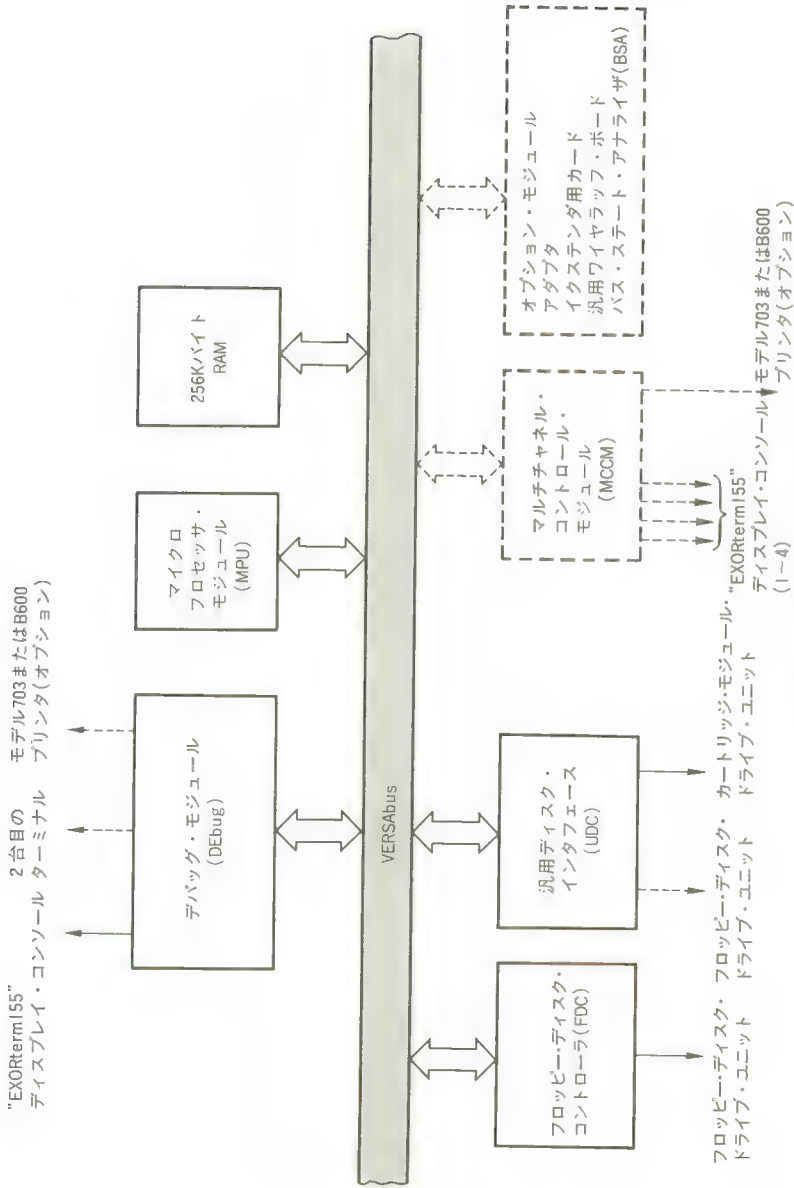


図 10・3 機能別ブロック・ダイアグラム

10 MC 68000 開発装置

システム用や、すべてのオペレーションが制限なしに実行できる**スーパーバイザ・モード**と、ユーザの応用プログラム用や、いくつかの特権命令の実行に制限のある**ユーザ・モード**です。

(2) **二重マップ方式**は、11・1 節の VERSAdos ソフトウェア参照。

(3) **バスの調停**は、マルチ・プロセッサ方式を実行できます。

(4) **インテリジェント・ペリフェラル・コントローラ (IPC)**により、オペレーティング・システムが現在のタスクを I/O 機器の制御に時間をとられることなく実行できます。

10・2 エクサマックスの使用例

例 COMPIL.CF

```
=ARG
=PASCAL \1/\2,,\3
=PASCAL2 \1,,#NULL
=LINK \1,,\1;IXM
=END
```

Arg1,2 で示される PASCAL・ソース・プログラムをコンパイルしてリンクの後、ロード・モジュールを作り出すもの

(実行)

```
=BATC COMPIL QUEENS,NULL;#PR ←一括モード処理の開始を指定する
OOOB: QUEUED
=BATC COMPIL QUEENS,NULL;#PR ←さらに同じ内容の処理を開始させる
OOOC: QUEUED
=QUER ←"QUER" コマンドによりシステム内のジョブの状態をチェックする
OOOB: RUNNING
OOOC: QUEUED
=CANC ←最後に登録したジョブを取り消す
OOOC: DONE=A006
=QUER ←"QUER" コマンドにより最後に登録されたジョブの状態をチェックする
OOOC: DONE=A006
=
```

この一括処理を行った結果として次のファイルが作成される

```
SYS:0000..QUEENS.CF
SYS:0000..QUEENS.SA
SYS:0000..QUEENS.PC
SYS:0000..QUEENS.RO
SYS:0000..QUEENS.LO
SYS:0000..QUEENS.LL
```

作成されたファイル

図 10・4 コマンド・ファイル (CDF) の内容

```
=ARG
=PASCAL \1/\2,,\3
=PASCAL2 \1,,#NULL
=/ABT
=LINK \1,,\1;IXM
=END
END CHAIN
```

ここで使用されるチェーン・ファイル
COMPIL1.CFは左の通り

```
=CHAI COMPIL1 QUEENS,NULL,#CN00 } COMPIL1をチェーン処理で実行する  
このとき、引数1,2,3をQUEENS,  
NULL,#CN00と指定する
```

```
=ARG
\1:QUEENS
\2:NULL
\3:#CN00
=PASCAL QUEENS/NULL,,#CN00
Motorola Pascal Compiler Phase 1 Version 1.20
Copyrighted 1981 by Motorola, Inc.
```

```
**** No Error(s) detected in compilation ****
=PASCAL2 QUEENS,,#NULL
M68000 Pascal Compiler Phase 2 Version 1.20
Copyrighted 1981 by Motorola, Inc.
```

```
CODE GENERATOR PRODUCED 000003C2 BYTES OF CODE.
NO ERRORS DETECTED.
=/ABT
RX=$0000 RA=$666C RD=$0000
CHAIN ABORTED
(CHAIN)=END
```

図 10・5 チェーン・モード例

IO MC 68000 開 発 装 置

"EXORmacs" システム生成用ファイル SYSCMD.SA

(1)

MSG SYSGEN COMMAND FILE FOR HARD DISK, MULTI-USER CONFIGURATION

MSG DEFINE ALL DEVICES

TOTTERM=10 TOTAL NO. OF TERMINALS

TOTPR=3 TOTAL NO. OF PRINTERS

TOTDSK=8 TOTAL NO. OF DISK DRIVES

NOLTERM=2 NO. OF LOCAL TERMINALS

NOLPR=1 NO. OF LOCAL PRINTERS

HDUDC0=4 NO. OF HARD DRIVES ON 1ST UDC

FDUDC0=2 NO. OF FLOPPY DRIVES ON 1ST UDC

HDUDC1=0 NO. OF HARD DRIVES ON 2ND UDC

FDUDC1=0 NO. OF FLOPPY DRIVES ON 2ND UDC

NOFDC0=2 NO. OF FLOPPY DRIVES ON 1ST FDC

NOFDC1=0 NO. OF FLOPPY DRIVES ON 2ND FDC

NOIPCS=2 NO. OF DISK FDC'S AND UDC'S

NOTERM0=4 NO. OF TERMINALS ON 1ST MCCM

NOPRT0=1 NO. OF PRINTERS ON 1ST MCCM

NOTERM1=4 NO. OF TERMINALS ON 2ND MCCM

NOPRT1=1 NO. OF PRINTERS ON 2ND MCCM

NOTERM2=0 NO. OF TERMINALS ON 3RD MCCM

NOPRT2=0 NO. OF PRINTERS ON 3RD MCCM

NOTERM3=0 NO. OF TERMINALS ON 4TH MCCM

NOPRT3=0 NO. OF PRINTERS ON 4TH MCCM

NOMCCMS=2 NO. OF MCCM'S

MSG DEFINE OTHER VARIABLE INFO.

NOTASKS=30 MAX. NO. OF TASKS IN SYSTEM AT ONE TIME

MAXLU=8 MAXIMUM LU FOR EACH TASK

DQQPGE=2 NO. OF PAGES IN DEVICE CONNECTION QUEUE

GST=4 NO. OF PAGES IN GLOBAL SEGMENT TABLE

UST=1 NO. OF PAGES IN USER SEMAPHORE TABLE

TRACE=2 NO. OF PAGES IN TRACE TABLE

IOV=1 NO. OF PAGES IN I/O VECTOR TABLE

NOFILES=50 MAX. NO. OF FILES OPEN AT ONCE

NOIFFIL=50 MAX. NO. OF DIFFERENT FILES OPEN AT ONCE

MODEFVOL=30 MAXIMUM NO. OF DEFAULT VOLUMES

DEFFAB=1 DEFAULT FAB LENGTH

DEFDAT=4 DEFAULT DATA BLOCK LENGTH

MMU=\$FE2000 MMU ADDRESS

TIMER=\$FEE040 TIMER ADDR.

PANEL=\$FE0000 FRONT PANEL ADDR.

MEMEND=\$180000 MAXIMUM END OF CONTIGUOUS RAM

MSG CHECK PARAMETER VALIDITY

SUBS VALPAR

ASM VALPAR, ,#CNOO; -C

PAUSE ABORT SYSGEN IF ANY ERRORS IN ASSEMBLY

MSG GENERATE EXEC PROCESS

PROCESS RMS

STACK=\$900 EXEC STACK AREA

STARTRMS=\$C00 EXEC BEGINNING ADDR.

RMSFATAL=\$C0E FATAL SYSTEM ERROR ADDR.

END EXEC

MEMBEG=* START OF AVAILABLE MEMORY

SUBS IXR

ASM IXR, IXR, #CNOO

MSG SET UP FHS TASK

TASK FHS, FHS

STATE='DORM'

SESSION=1

任意のメッセージ
を出力することが
できる

パラメータを設定
することができる

アセンブラを実行
することができる

タスクとプロセスの処理を指定することができる

④ キーボードからの入力待つこともできる

図 10・6 SYSGEN 例

(2)

```

PRIORITY=$D1
FHSSTR=* FHS LOAD ADDR.
FHSASR=**+2 FHS ASR ENTRY POINT
SUBS FHSLNK.CF
LINK FHSLNK ← IOSLNK.CF
END FHS
MSG START OF IOS TASK
TASK IOS,.IOS
PRIORITY=$D1
IOSSTR=* IOS LOAD ADDR. .....①
IOSASR=**+2 IOS ASR ENTRY POINT .....①
SUBS IOSLNK.CF .....①
LINK IOSLNK .....②
END IOS
MSG START OF IOD TASK
TASK IOD,.IOD
PRIORITY=$D1
IODSTR=* IOD LOAD ADDR.
IODASR=**+2 IOD ASR ENTRY POINT
SUBS IODLNK.CF
LINK IODLNK
END IOD
MSG START OF TTY TASK
TASK TTY,.TTY
PRIORITY=$D4
TTYSTR=* TTY LOAD ADDR.
TTYASR=**+2 TTY ASR ENTRY POINT
SUBS TTYLNK.CF,TTYB
ASM TTYB,TTYB,#CNOO
LINK TTYLNK
END TTY
MSG START OF PRT TASK
TASK PRT,.PRT
PRIORITY=$DB
PRTSTR=* PRT LOAD ADDR.
PRTASR=**+2 PRT ASR ENTRY POINT
SUBS PRTLNK.CF,PRTB
ASM PRTB,PRTB,#CNOO
LINK PRTLNK
END PRT
MSG START OF IPC TASK
TASK IPC,.IPC
PRIORITY=$D2
IPCSTR=*IPC LOAD ADDR.
IPCASR=**+2 IPC ASR ENTRY POINT
SUBS IPCLNK.CF,IPCB
ASM IPCB,IPCB,#CNOO
LINK IPCLNK
END IPC
MSG START OF COM TASK
TASK COM,.COM
PRIORITY=$D4
COMSTR=* COM LOAD ADDR
COMASR=**+2 COM ASR ENTRY POINT
SUBS COMLNK.CF,COMB
ASM COMB,COMB,#CNOO
LINK COMLNK ← LINKER を実行させることができる

```

①においてIOSLNK.CFの“IOSSTR”にSYSGENユーティリティにより管理されている位置カウンタの値が与えられている
位置カウンタの値は、①においてパラメータコマンドにより求められている

図 10・7 IOS TASK 例

(3)

```

END COM
MSG START OF FMS
TASK FMS,.FMS
PRIORITY=$DO
FMSSTR=* FMS LOAD ADDR.
FMSASR=**+2 FMS ASR ENTRY POINT
SUBS FMSLNK.CF
LINK FMSLNK
END FMS
MSG START OF EET
TASK EET,&EET
STATE='READ' ←
SESSION=2 ←
PRIORITY=$C8
EETSTR=* EET LOAD ADDR.
SUBS EETLNK.CF
LINK EETLNK
END EET
MSG START OF LDR
TASK LDR,&LDR
PRIORITY=$C8 ←
SESSION=4 ←
LDRSTR=* LDR LOAD ADDR
SUBS LDRLNK.CF
LINK LDRLNK
END LDR
MSG START OF IOCOM AND IOI
TASK IOI,.IOI
SESSION=1 ←
PRIORITY=$DA ←
IOCSTR=*
SUBS IOC
ASM DVM/IOC,IOC,#CN00
SUBS ASR
ASM ASR,ASR,#CN00
SUBS IOILNK.CF
LINK IOILNK
END IOI
MSG START OF SYSTEM INITIALIZER
PROCESS INT
INTSTR=*
SUBS IND,INTLNK.CF
ASM IND,IND,#CN00
LINK INTLNK
END INT

```

パラメータにはシステム生成ユーティリティにより使用されているものもある

これらは初期値が設定されている
STATE('READ')
SESSION(0)

```

=LINK,IOS=#PR,IXHM
SEGMENT .IOS:0 \IOSSTR
INPUT IOS
END
=END

```

AS	ハードウェアを使用したアドレス・ストップ条件を設定します。
BD	ディスク媒体へのメモリ内容のダンプを行います。
BH	ディスク内のデータをメモリにロードし、"MACSbug" にコントロールを移します。
BI	メモリ・ブロックの非破壊的初期化を行います。
BO	セクタ 0 の情報によりデータのブート・ロードを行います。
BR	ブレーク・ポイント・テーブルへのアドレスの設定を行います。
BT	メモリ・ブロックの破壊的テストを行います (0 に初期設定されます)。
CA	ユーザ作成コマンドへのリングを行います。
DC	16 進、10 進への交換を行います。
DF	トレース表示形式の指定を行います。
DU	メモリの内容を S 形式で出力します。
GO	プログラムの実行を指定します。
GT	ブレーク・ポイント (一次的) までのプログラムの実行を指定します。
HE	有効なコマンドの情報を与えます。
LO	S 形式のオブジェクト・データを外部デバイスからメモリへロードします。
MD	メモリの内容を表示します。
MM	メモリの内容を変更します。
MS	メモリの内容を設定します。
NOAS	"AS" によりセットされた条件を解除します。
NOBR	"BR" によりセットされた条件を解除します。
NODEF	トレースにおける表示形式から一つ以上の項目を削除します。
NOPA	ライン・プリンタ出力を禁止します。
OF	リロケータブル・ファイルのオフセットを表示します。
PA	ライン・プリンタ出力を可能にします。
PF	おのおののポートの特性を設定します。
PR	"DEbug" モジュール・ハードウェアをプライマリ・マップ (一次マップ) に設定します。
RM	各レジスタの内容を変更します。
SE	"DEbug" モジュール・ハードウェアをセコンダリ・マップ (二次マップ) に設定します。
T or TR	プログラムの実行を 1 インストラクションごとに行います。
TM	HOST から直接アクセスできるように "DEbug" モジュールの ACIA ポートを接続します。 (PORT1 と PORT2 は同じボーレート)
TT	一次的なブレーク・ポイントまでトレースします。
VE	外部デバイスからの S 形式のオブジェクト・データと現在のメモリ内容を比較します。
※	ポート 2 にメッセージを転送します。
・Aφ--A7	Aφ-A7 の内容を表示又設定します。
・Dφ--D7	Dφ-D7 "
・Rφ--R6	Rφ-R6 "
・PC	PC "
・SR	SR "
・SS	SS "
・US	US "

図 10・9 MACSbug 命令一覧

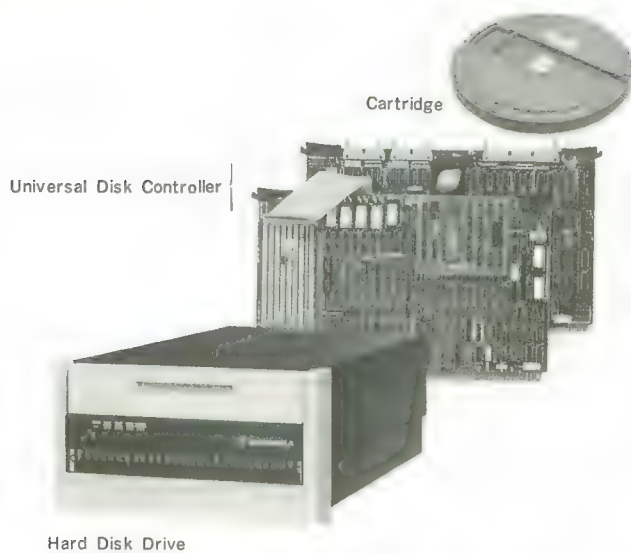


図 10・10 M 68KHDS32 基本モジュールの構成

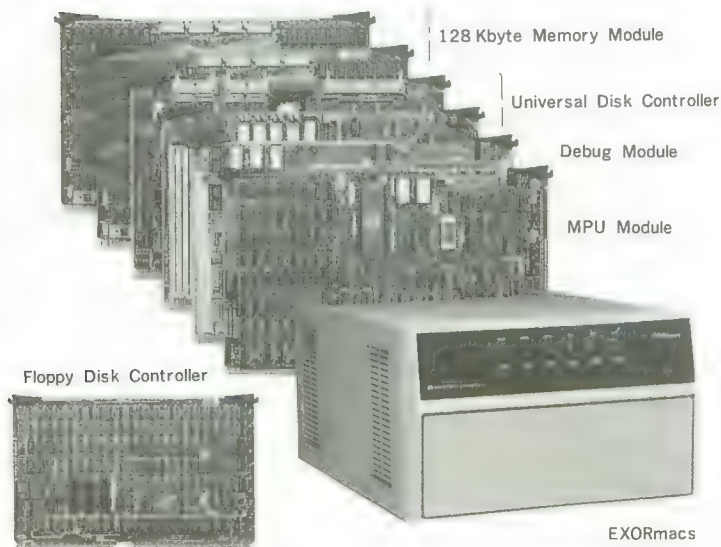


図 10・11 EXORmacs 基本モジュールの構成

10・3 VMC 68/2 マイクロコンピュータ

VMC 68/2 は、アプリケーションの基本要素をすでに実装済みで、ユーザが自分の仕様に合わせたボードを選択して追加し、アプリケーション・プログラムを VMC 68/2 上で作成し、デバッグ、シミュレート、不要要素を除いてシステム・ソフトウェアに組み入れることで、リアルタイム・マルチタスクのアプリケーションを完成させ、そのまま実際の応用機器に使用できます。したがって VMC 68/2 は、多様な仕様に合致させられるような仕様であり、かつ開発期間を短縮することのできる組み込み用 OEM 向けの開発支援装置です。

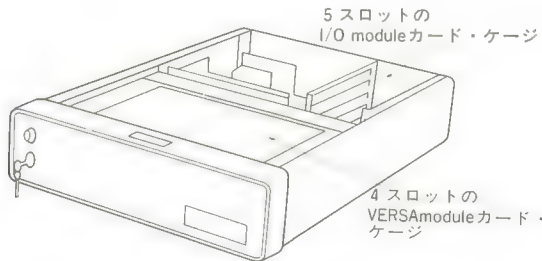


図 10・12 VMC 68/2 MC 68000 マイクロコンピュータ・システム

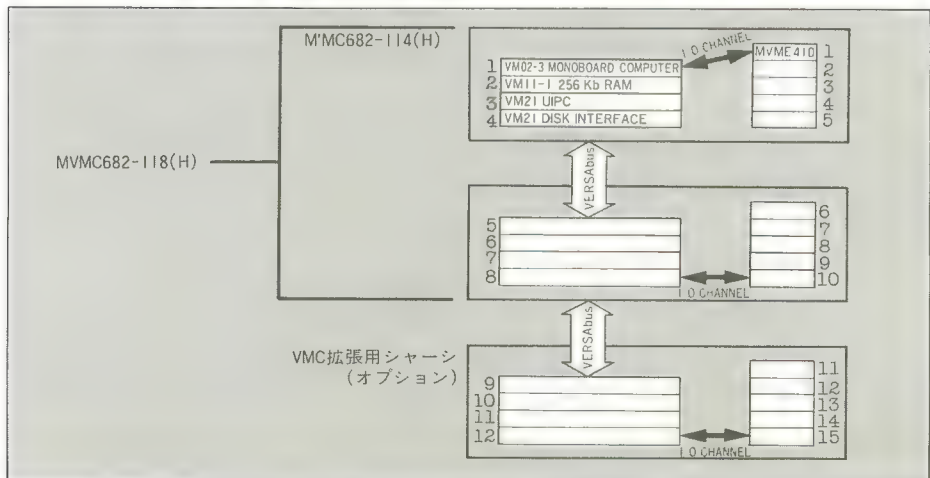


図 10・13 VMC 68/2 の構成

10 MC68000 開発装置

VMC 68/2 の構成は、次の三つの開発要素から構成されています。

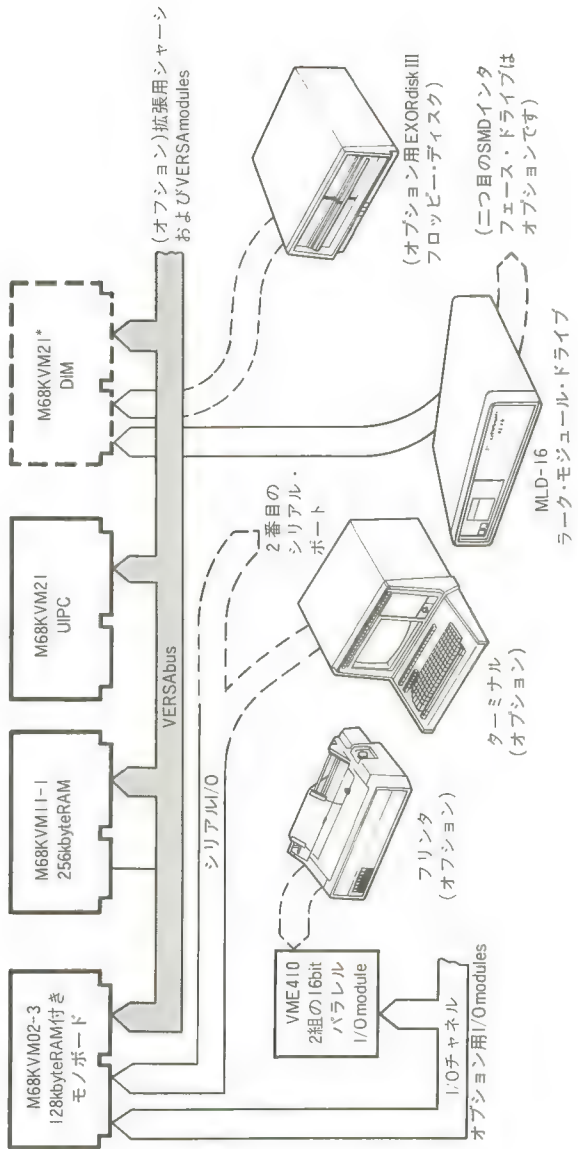
- (1) 処理装置と内蔵基本ボードなどの基本要素
- (2) VERSA module, I/O Module, パッケージ・ハードウェアなどの拡張要素
- (3) エディタ, アセンブラ, リンカ, デバッガ, システム生成ユーティリティとファイル装置などの開発要素

VMC 68/2 の特長は、次の4点が代表的なものです。

- (1) MC 68000 (16/32 ビット MPU) を搭載し, UERSAdas リアルタイム・マルチタスキング・オペレーティング・システムにより高い制御能力を持ちます。
- (2) メモリ・マップド I/O 方式により, I/O, メモリ, 大容量ファイル装置などの拡張が容易です。
- (3) 34 K バイトの内蔵メモリを備え, 16 M バイトの直接アドレス空間を持ちます。
- (4) デバッガ, 単一ライン逆アセンブラ, 自己診断, ディスク・ブート・アップ/ダウン・ロードなどのパーサバグ (VERSAbug) のファームウェア。

VMC 68/2 の基本構成は, VERSA モジュール・ファミリから構成されており, VMC 68/2 へ豊富な拡張性を与えています。つまり, VESA モジュール・ボード・ファミリや I/O モジュール・ボード・ファミリなどのメモリや I/O, 周辺制御モジュール, そして, 他のメーカーから供給される VERSA バス仕様と互換性のある製品に拡張用ハードウェアを VMC 68/2 へ追加することにより, 簡単に拡張することができます。VMC 68/2 のメモリや I/O 関係のシステム概要を図 10・14 に示し, 図 10・15 に VMC 68/2 の使用におけるハードウェアとソフトウェアの構成セレクション・チャートを示し, 図 10・16 に VMC 68/2 の VERSA モジュール・カードと I/O モジュール・カードの構成を示します。

ソフトウェアの対応は, ハードウェアの拡張で追加されたハードウェアに対応するプログラムをモジュラー形式で作成し, 次に **SYSGEN** ユーティリティを利用することによって実現できます。また, 容易に VERSA ドスに組み込むことができます。そして, カスタム・デバイス・ドライブのソフトウェアの追加も簡単に行うことができます。



* 注意: VMC68/2のカード・スロットをフリーにするため、オプション・マウント・キット (M68RDIMK) 付きのMLD-16ディスク・ドライブにM68KVM21DIMを取り付けることができます。

図 10-14 VMC68/2のシステム概要

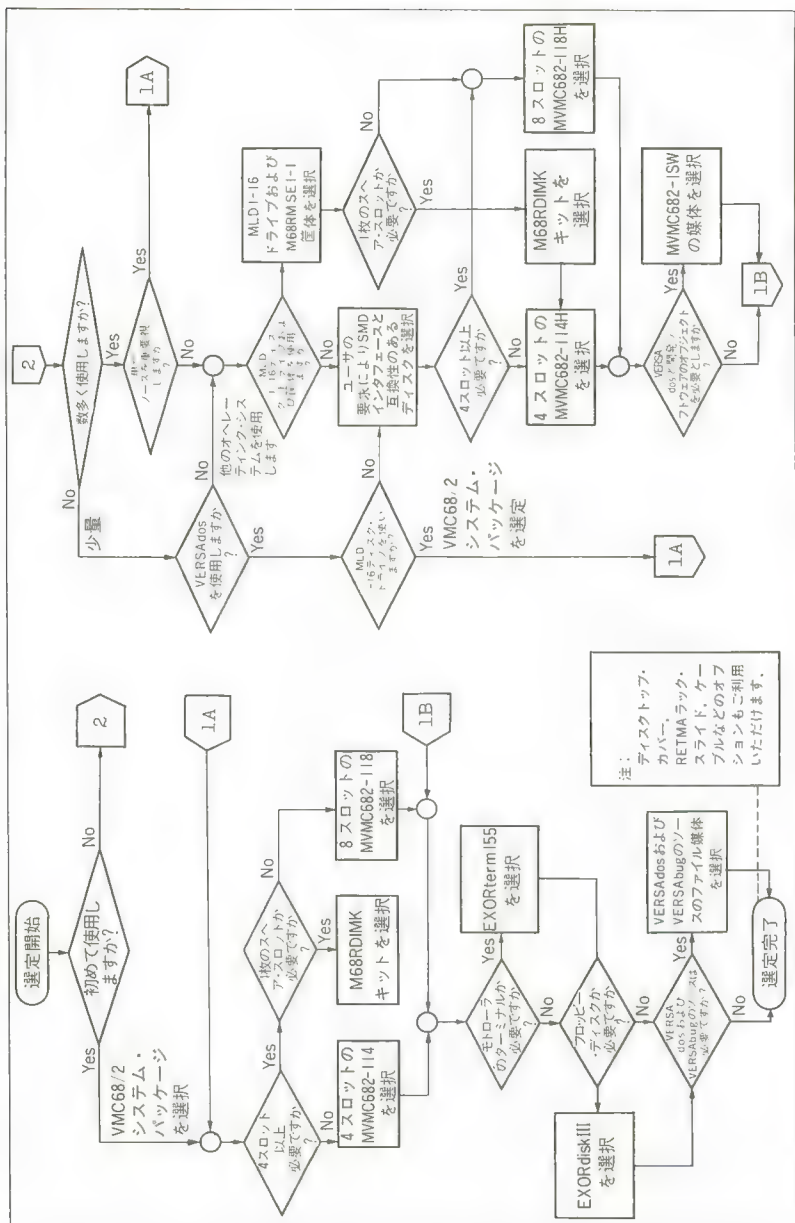


図 10・15 VMC 68/2 セレクション・チャート

例として、VMC 68/2 の演算制御用の VERSA モジュールのモノボード・コンピュータ (M68 KVM 02) がありますが、このマイクロコンピュータ・モジュールは、全般的な機能の作成および周辺機器制御の要求に応ずるために、I/O チャネル・インタフェースを備えています。そして I/O チャネルは、最高 15 枚の I/O モジュールを取り付けることができます。そして、この M68 KVM 02 は、128 K バイトの双方向 RAM を持ち、マルチプロセッサ構造を備えています。

システム・パッケージは、MLD-16 ディスク・ドライブと VERSA ドス・リアルタイム・マルチタスキング・オペレーティング・システムを含みます。VERSA ドス・オペレーティング・システムは、OEM あるいはシステム作成者が最終的に使用する VERSA ドス構造のアプリケーションを作り上げるために使用するシステム生成ユーティリティを持ちます。この OS は、フォワードとバックグラウンドでバッチ処理ができ、コマンド・プロセッサは、コマンドのチェーンでファイルからのコマンド実行、待ち行列タスクの実行ができます。

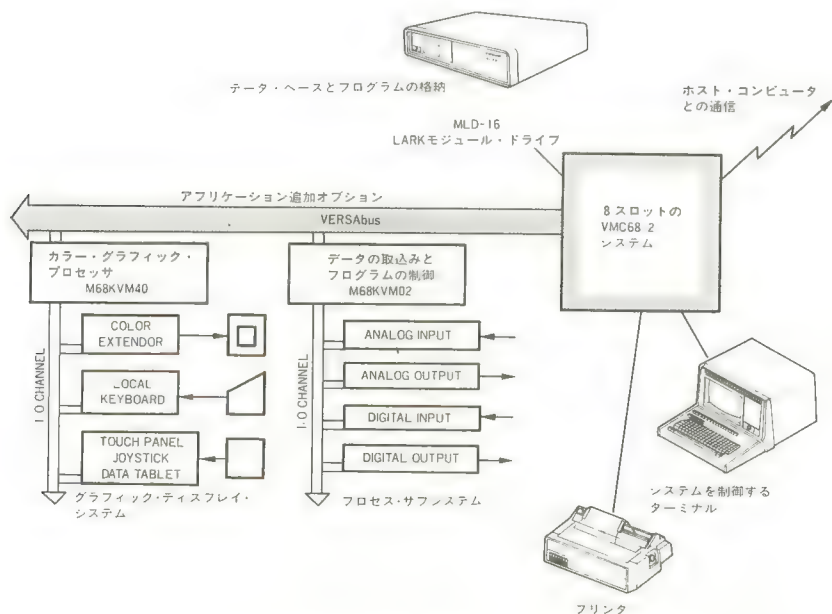


図 10・16 VMC 68/2 を中心として拡張用モジュールを追加したアプリケーション

IO MC 68000 開 発 装 置

VMC 68/2 の拡張用 VERSAmodule

製 品 名	説 明
M 68 KVM 02-3	128 Kbyte の双方向 RAM 付きモノボード・マイクロコンピュータ
M 68 KVM 10-2	64 Kbyte RAM モジュール
M 68 KVM 10-3	128 Kbyte RAM モジュール
M 68 KVM 11-1	ECC 付き 256 Kbyte RAM モジュール
M 68 KVM 11-2	ECC 付き 512 Kbyte RAM モジュール
M 68 KVM 21	ユニバーサル・ディスク・コントローラ (UDC)
M 68 KVM 20	フロッピー・ディスク・コントローラ (FDC)
M 68 KVM 30	4-チャンネル・コミュニケーション・モジュール (MCCM)
M 68 KVM 31*	8-チャンネル・コミュニケーション・モジュール
M 68 KVM 40	カラー・グラフィック・プロセッサ・モジュール
M 68 KVM 60	ユニバーサル・インテリジェント・ペリフェラル・コントローラ (UIPC)
M 68 KVM 80-1	RAM なし、メモリ、I/O、時計付きコンビネーション・モジュール
M 68 KVM 80-4	128 Kbyte RAM、メモリ、I/O、時計付きコンビネーション・モジュール

* 計画中の製品です

拡張用 I/O module

製 品 名	説 明
M 68 RIO 1	16 I/O ポイントのソリッド・ステート・リレー I/O module
M 68 RAD 1	インテリジェント A/D 変換 I/O module
MVME 400	2 組の RS-232 C シリアル I/O module
MVME 410	2 組の 16 bit パラレル I/O module
MVME 420	SASI インタフェース・アダプタ I/O module
MVME 435*	9-トラック、1/2 inch マグネティック・テープ・アダプタ I/O module
M 68 RWIN 1*	5-1/4 inch と 8 inch ウィンチェスタおよび FDC コントローラ I/O module
M 68 RFDC 1*	5-1/4 inch と 8 inch フロッピー・ディスク・コントローラ I/O module
M 68 RSC 1	RS-232 C ~ RS-422 のマルチドロップのシリアル・コンバージョン・モジュール
M 68 RSC 2	RS-232 C ~ RS-422 のターミナル・アダプタ

* 計画中の製品です

11. 開発ソフトウェア/ リアルタイム・モニタ

従来のアセンブラは、生産性や記述性やドキュメント性が低く、柔軟性にも欠けていた。この点を補う方法として、上位言語が用いられるようになってきました。68000 VERSAdos 開発ソフトウェアに含まれる標準ソフトウェアの種類と内容を説明し、開発装置の特長の一つである二重マップ方式を説明します。また、RMS 68 K マルチタスキング・ソフトウェアは、システム支援の中心的機能ともいえるタスク管理の種類と機能について説明します。

11.1 VERSAdos 開発ソフトウェア

VERSAdos は、標準開発ソフトウェアとして用意した設計者の機器開発のツールです。標準装備のソフトウェアとして、CRT エディタ、PASCAL コンパイラ、ストラクチャード・マクロ・アセンブラ、シンボリック・デバッガ・リンケージ・エディタがあります。オプションとしてのプログラムは、FORTRAN, MPL, BASIC-M, COBOL, C/UNIX, ADA, ADLFORTH などの言語をサポートします。

図 11.1 に開発サポートの基本構成を示します。

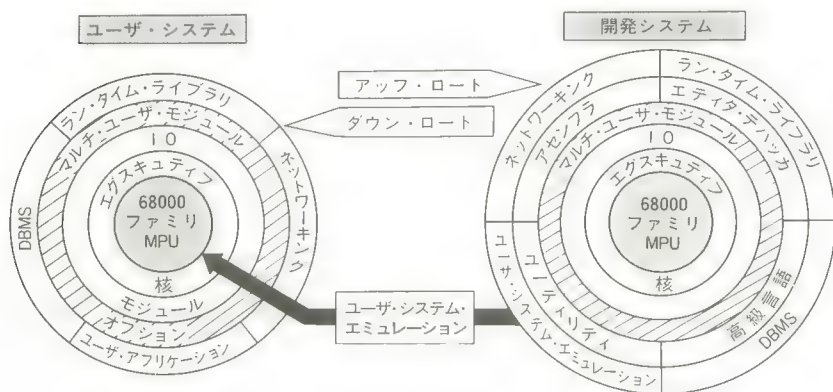


図 11.1 ユーザ・システム・アプリケーション開発装置

CRT エディタは、オペレーティング・システムの監督下で働き、CRT 機能を有効に使用し、CRT 内部でソース・プログラムを効率良く作成や修正ができます。

ストラクチャード・マクロ・アセンブラは、ソース・ステートメントをリロケータブル・マシン・コードに変換し、メモリ・ロケーションを命令およびデータに割り当て、プログラマが指定したアセンブラ作業を行い、随意にクロス・リファレンス・リストを作ります。また、ストラクチャード・プログラミング構成に加えて、マクロおよびコンディショナル・アセンブリの機能も備えています。

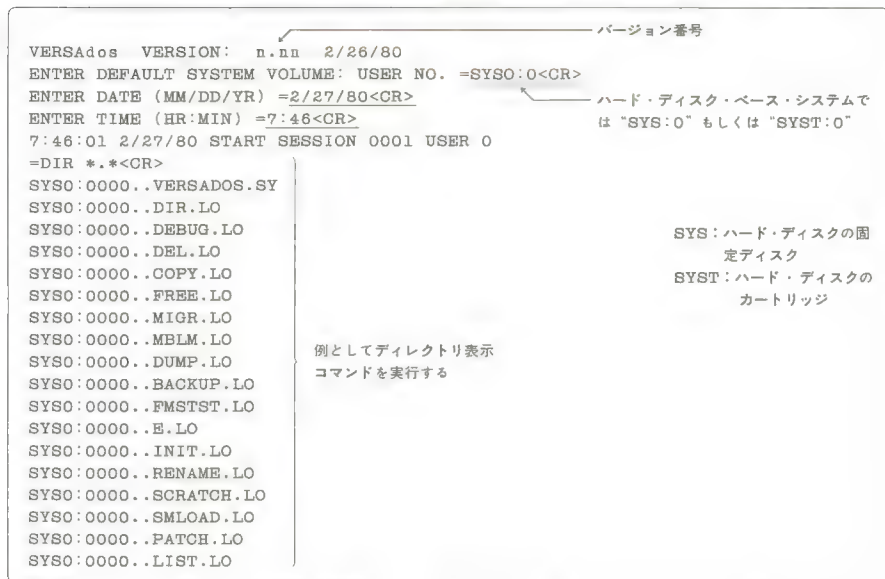


図 11・2 VERSAdos の例

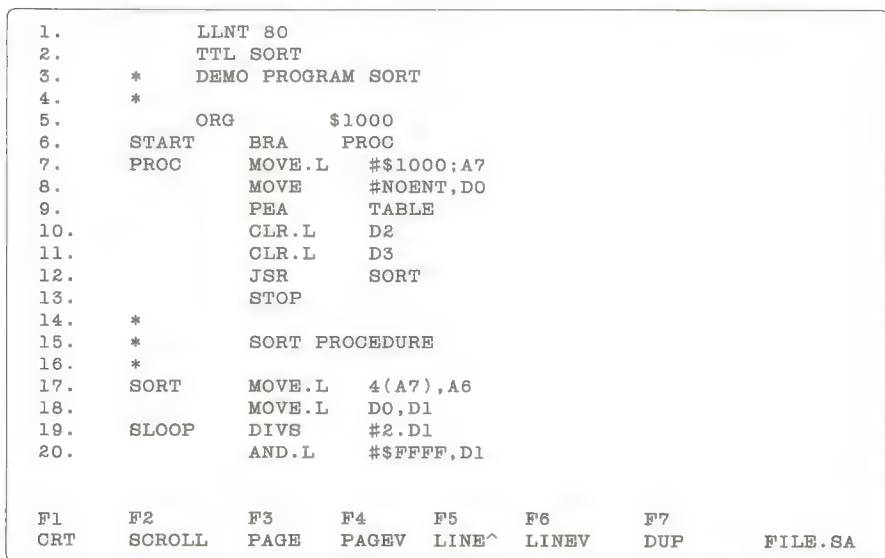


図 11・3 CRTエディタ

11 開発ソフトウェア/リアルタイム・モニタ

PASCAL コンパイラは、プログラミング技術や手法を促進する高度に構造化された言語であり、プログラム記述の容易さやドキュメント性に優れており、プログラムの製品性を高めます。

FORTRAN コンパイラは、ANSI, FORTRAN ワクランゲージの仕様をしのぐものであり、ISA イクステンションと互換性のあるリアルタイム・プロセッシング機能を備えています。そして ROMable で、リロケートブルなオブジェクト・コードを作成します。

リンケージ・エディタは、コンパイラやアセンブラなどの異なった言語を使用して出力した個々のリロケートブル・オブジェクトを結合し、一つのロード・モジュールを作ります。この作業において、セグメント・アトリビュートの指定、アドレス空間の計算、ライブラリの検索、外部参照ラベルの定義、オブジェクト・

```
* TRACE 3 ASSEMBLY INSTRUCTIONS
$001000 43F81026          LEA      DATA, AI
$001004 2049             MOVE.L  AI, AO
$001006 2610             MOVE.L  (AO), DO

* DISPLAY MEMORY      ?AI ?AO:2 (AO):4 ?DO:3
?AI=$00001026 ?AO=$04134 (AO)=$000076A9 ?DO=$0076A9

* DISPLAY SYMBOL      DATA
DATA=$1026
```

図 11・4 シンボリック・デバッグ (ASSEMBLER)

```
* TRACE 1 PASCAL INSTRUCTION
IF long[I] > 255 THEN

* SET MEMORY long[I] 260

* TRACE 1 PASCAL INSTRUCTION
FOR i := I TO 4 DO

* TRACE DISPLAY i long[i] carry temp

* TRACE 3 PASCAL INSTRUCTIONS
temp := long[i] + carry;      i=I long[i]=260 carry=0 temp=260
carry := temp DIV 256;        i=I long[i]=260 carry=I temp=260
long[i] := temp MOD 256       i=I long[i]=4 carry=I temp=260
```

図 11・5 シンボリック・デバッグ (PASCAL)

コードのリロケート，エラー・メッセージの表示を行います。さらに，モジュール・マップや外部定義シンボル・テーブル，未定義や二重定義シンボルをレポートします。

シンボリック・デバッガは，ランゲージ・プロセッサやリンケージ・エディタで作成されたプログラムを，SYMbug によってステートメント・レベルでデバッグします。ステートメント，数値，ブロックのようなプログラム・エレメントをシンボリックに調べて修正します。ブレイク・ポイントのプログラムへの挿入やブレイク要求，あるいはデータの変更を含めて，シンボリックに実行をコントロールします。特殊なエレメントとサブ・エレメントをシンボリックに検索します。ユーザ定義のマクロ・コマンドの作成ができ，指示されたプログラム位置からトレースが実行できます。

```

M68000 ASSEMBLY LANGUAGE PROGRAM WHICH FINDS
THE GREATEST VALUE IN ARRAY,DATA , LEAVING
THAT VALUE IN REGISTER D3 AND THE ADDRESS OF
ITS FIRST OCCURRENCE IN REGISTER A0

ORG      $1000
OPT      BRS,FRS                      SHORT FORWARD REFERENCE OPTIONS
LEA      DATA,AI                     LOAD BASE ADDRESS INTO AI
MOVE.L   AI,A0                        A0 STARTS WITH FIRST ELEMENT
MOVE.L   (A0),D3                      D3 STARTS WITH FIRST ELEMENT

FOR.L DI=#4 TO #4*(SIZE-1) BY #4 DO   : DI IS USED AS AN INDEX REG
    IF.L (AI,DI) GT D3                 : SEARCH IS SUCCESSFUL
        THEN BEGIN
            LEA      (AI,DI),A0        : LOAD NEW ADDRESS
            MOVE.L   (A0),D3           :LOAD NEW VALUE
        END
END

*
SIZE     DC          $20
DATA     DS.L        SIZE
END
    
```

図 11・6 ストラクチャード・マクロアセンブラ

11 開発ソフトウェア/リアルタイム・モニタ

ソフトウェア開発における全体の流れ図を、図 11・7 に示します。

開発用アプリケーション・プログラムの使用方法として、M 68000 EXORmacs 開発システムを用いて、M 68000 CRT エディタにてアセンブリやデバッグングを行ないます。

文字の入力は、EXORterm 155 キイボードを用います。また、プログラム・ファイルの入力には、ディスク上の EXORmacs オペレーティング・システムである VERSAdos より行ないます。

EXORmacs 開発システムは、SYMbug、DEbug、MACSbug の 3 種類のモニタ/デバッグを備えており、それぞれプログラムのローディングやメモリのテストおよびプログラムの実行ができます。また、MACSbug は、シングル・ユーザのみに用いられますが、SYMbug と DEbug は、VERSAdos のもとにマルチ・ユーザにて用いることができます。

デバッグングのフォーマットは、次のようになっています。

MACSbug コマンド・フォーマット

P*[No] <コマンド> [<パラメータ>] [; <オプション>]

SYMbug コマンド・フォーマット

SYMbug [<タスク>] ? [No] <コマンド> [<パラメータ>] [; <オプション>]

DEbug コマンド・フォーマット

DEbug [<タスク>] ? [No] <コマンド> [<パラメータ>]

VERSAdos コマンド・フォーマット

= <コマンド> [<入力フィールド>] [, <出力フィールド>] [; <オプション>]

EXORmacs の二重マップ方式

EXORmacs のメモリ・マップは、図 11・8 に示すように、メモリ・マップは一次と二次の二重マップ方式をとっています。一次マップは、プログラムはプログラム開発用エリアとし、二次マップは応用プログラムのエミュレーションのエリアとして、EXORmacs システムのエミュレーションや開発能力へのフレキシビリティを持たせ、最終的な設計や開発において拘束されることなく実行できることです。

デバッグ方法は、図 11・8 に示すように **SMLOAD** プログラムをロード・モジュ

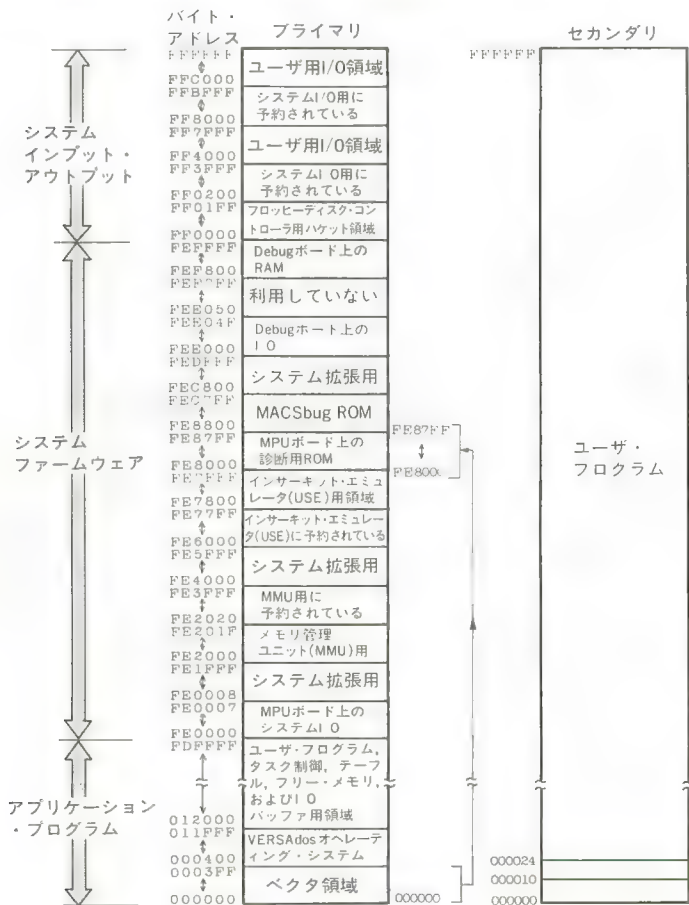


図 11・8 EXORmacs のメモリ・マップ

ールの二次マップ・メモリにロードし、MACSbugによってデバッグが行われます。

MACSbugは、一次マップでも二次マップでも実行が可能です。図11・9に、一次マップから二次マップへの切換えフロー図を示します。また、図11・10にメモリ・二重マップを示します。

二重マップ方式は、ユーザに対して“EXORmacs”のサポートおよびMC 68000マイクロプロセッサの制限のない使用手段を供給するためのものであり、一次マップにおけるすべての領域では、すべてのインストラクションが実行でき、どのメモリにもアクセスできます(\$10と\$24は除く)。また、二次マップ使用中は、一次マップに格納されているシステム・ソフトウェアおよびシステムI/Oは、ユーザのプログラムと完全に分離されます。

注：、印は、“CR”キーの入力を示す。

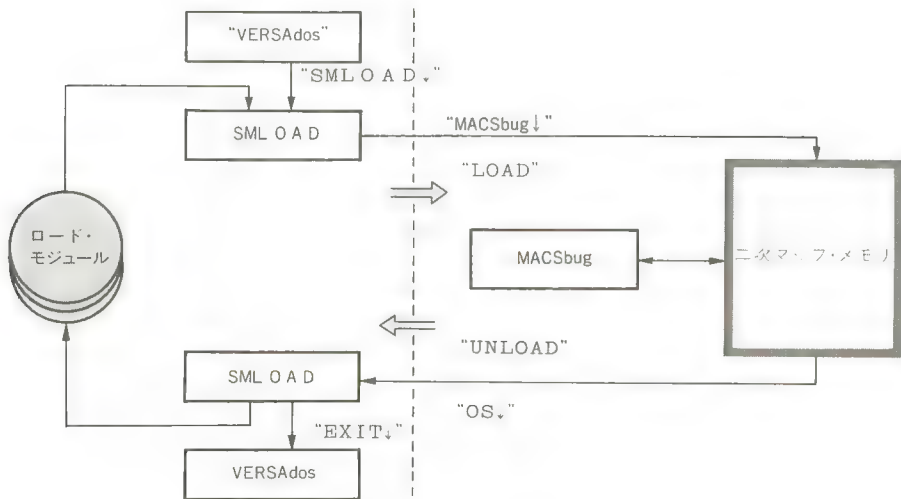


図 11・9 二次マップの実行フロー図

図 11・9 のフロー図補足

“SMLOAD” は、ロード・モジュールの LIB（ローダ・インフォメーション・ブロック）に従い、絶対アドレスとしてロードされますので、二次マップにおけるメモリの割当てや配置に考慮が必要です。

SMLOAD ファイル・コール：モジュールを二次マップにローディングする。

MACSBUG ↓ 入力：二次マップの “MACSbug” 状態に戻る。

UNLOAD コマンド：ロード・モジュールをライブラリのボリュームに格納。

OS ↓ 入力：“VERSAdos” 状態に戻る。

[注意] SMLOAD が目的のタスクを二次マップにロードされるのに使用されているときは、一次マップへは **PR** コマンドを使用しても受け付けられない。

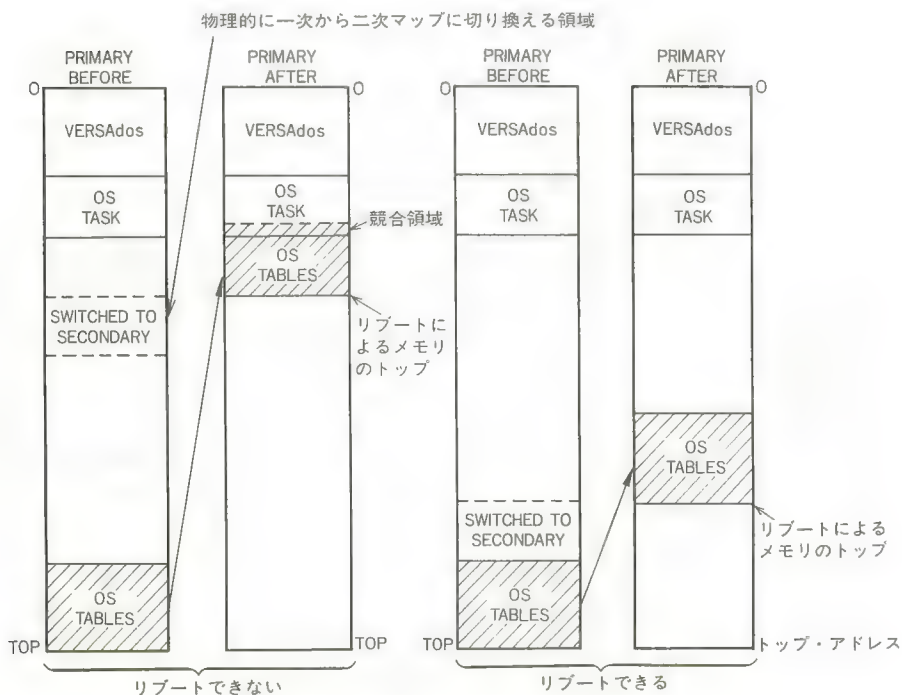


図 11・10 一次マップから二次マップへの切換え

11・2 RMS 68K リアルタイム・モニタ

RMS 68 K/VDOS は、表 11・1 に示すようなプロセッサ管理、メモリ/タスク管理、ファイル管理、I/O 管理とからの 4 部構成になっており、マルチタスキング・ソフトウェアです。リアルタイムで動作する応用機器やマルチユーザで動作する応用機器やアセンブラ、リンカ、エディタ、パスカルなどのサービス・プログラムを利用する応用機器のソフトウェアの核となっています。

システム・ソフトウェアの構成は、図 11・11 に示すような構成になっており、ユーザはユーザ・プログラムを作成し、EXORmacs の **SYSGEN** ユーティリティを使用してシステム・ソフトウェアと結合することによって、応用機器のプログラムを完成することができます。

特長として次の項目があげられます。

- (1) リアルタイムのマルチタスク動作であり、タスク間の同期を行える。
- (2) タスクの優先レベル (0~255) まで設定が可能であり、タスク数に制限を持たない。
- (3) モニタ・タスク機能とサーバ・タスク機能を持つ。
- (4) システム・ジェネレーションで固有のシステム構造が可能であり、ユーザの I/O デバイス・ドライバ機能記述が可能です。
- (5) ソフトウェアとハードウェアの割込みに対するディスパッチング機能。

表 11・1 システム・ソフトウェア



RMS: Realtime Multitasking Software
VDOS: OEM VERSAdos Operating System

11 開発ソフトウェア/リアルタイム・モニタ

リアルタイム・システムは、動作を逐次的に行なっていくバッチ・システムとは異なり、ある動作が終了しないうちに、ほかの動作を開始したり、続行したり終了したりすることができます。このようにリアルタイム・システムの並行処理機構は、あたかも、いくつかの動作が同時に実行されているかのように見えます。

RMS 68 K は、タスク制御機構、タスク間通信機構、メモリ管理機構、初期化機構の四つの機能より構成されています。

RMS 68 K は、図 11・12 に示すようにレベル 0～レベル 5 までの六つのレベル・モジュールから構成されており、レベル 3～5 は、タスクにより直接アクセスできるモジュールであり、レベル 0～2 は、レベル 3～5 のモジュールをサポートするモジュールです。

モジュールのメモリ必要量

RMS 68 K のメモリ構成は、レベル 0～4 の部分とレベル 5 の部分との二つの部分に分けることができます。

レベル 0～4：12 K バイトのプログラム・コード領域と 800 K バイトのデータ領域

レベル 5：4.5 K バイトのプログラム・コード領域

なお、RMS 68 K の供給は、5 枚の両面単密度ディスクセットや 96 M バイト・ハード・ディスクなどにより供給されます。

レベル 0：プロセッサ管理機能

タスクへの実行権の割当て機能と基本同期機能と例外処理機能および割り込み処理機能を含みます。

レベル 1：物理メモリ管理機能

レベル 2：ユーティリティ機能

レベル 3：タスクのアドレス空間とメモリ・セグメントの管理機能

レベル 4：タスクの生成、削除、実行制御機能

レベル 5：チャンネル用物理入出力機能（オプション）

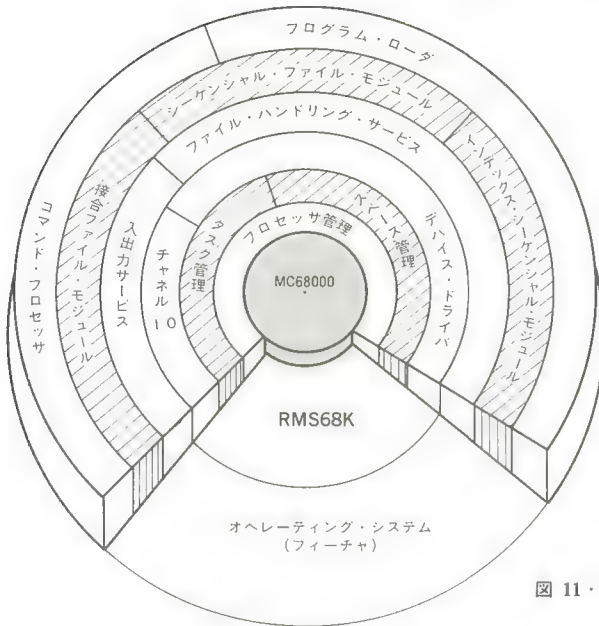


図 11・11 システム・ソフトウェアの構成

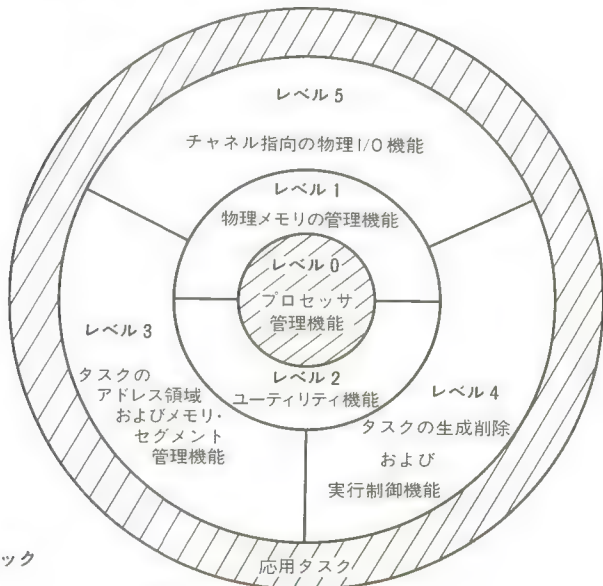


図 11・12 RMS68K の機能ブロック

11 開発ソフトウェア/リアルタイム・モニタ

応用システムのプログラムは、RMS68 Kのもとにタスクとして実行され、タスクは、タスクの優先度やスケジュールやタスクの要求資源の利用の可否に従ってスケジューリングされます。また、タスクは、機能的にまとまったある単位を実行するプログラムと、それに関連するデータ領域よりなります。

タスクのタイプとして、ユーザ・タスクとシステム・タスクの二つの基本的な型に分類でき、このいずれかに属します。

ユーザ・タスクは、通常の実用タスクであり、エグゼクティブ命令を用いて、メモリの割付け、解放、管理、他のタスクとの通信、タスク間の同期、タスクに従属する例外処理、物理入出力チャンネルのインターフェースなどの機能を実行することができます。

システム・タスクは、ユーザ・タスクに対してサービスを提供するために働くタスクであり、まったく制限なしに全資源をアクセスでき、すべてのエグゼクティブ命令を実行できます。

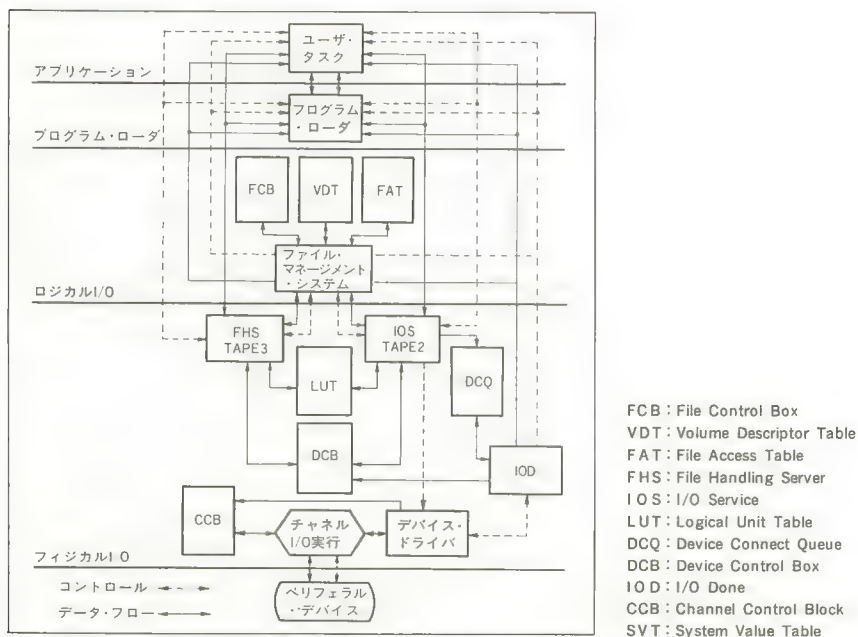


図 11・13 入/出力の要素

RMS 68 K はタスクにより制御され、それぞれのタスクに対してタスク制御ブロック (TCB) を備え、タスク状態制御用リンク情報、各テーブルのリンク情報、レジスタ退避領域などにより成り立っています。

タスクは、図 11・14 に示される状態を遷移することにより、動的に実行されます。セマフォは、タスク間の同期や複数のタスク間の共有データ領域の非同期な変更などに用います。また、セマフォの P/V オペレーションは、WTSEM と SGSEM の二つのディレクティブ命令によりサポートし、1, 2, 3 の 3 タイプのセマフォを持っています。

タイプ 1 は、複数のタスクによる一つの資源への排他的アクセスを実現します。

タイプ 2 は、タスクの順序処理を実現します。

タイプ 3 は、タイプ 1 の拡張であり、計数型のセマフォです。

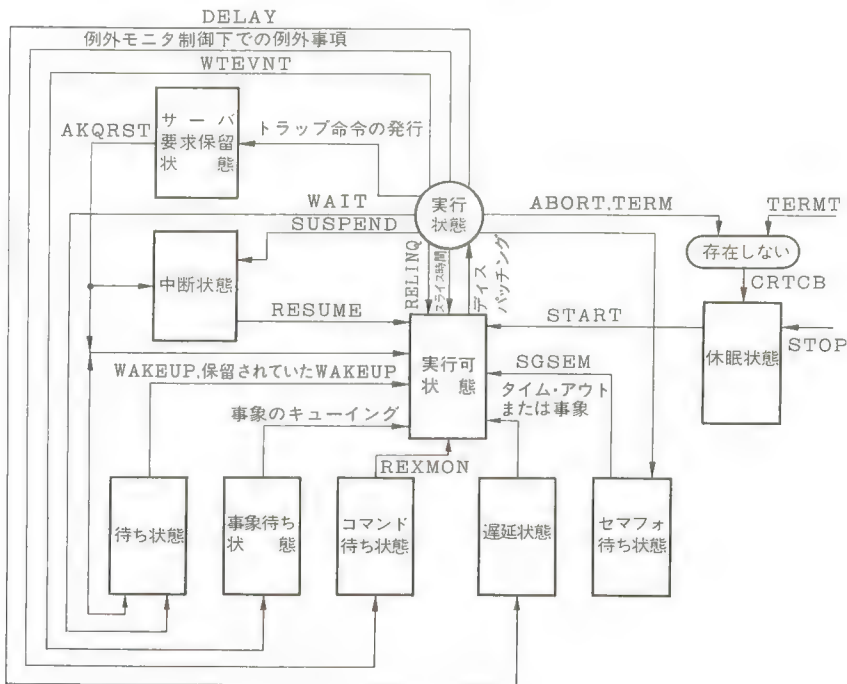


図 11・14 タスク状態遷移図

00000000	00000004	TCB	DS.L	1	'TCB' DUMP EYE CATCHER
00000004	00000004	TCBALL	DS.L	1	LINK FOR ALL TCB LIST
00000008	00000004	TCBGROUP	DS.L	1	LINK FOR TCB-THIS-GROUP LIST
0000000C	00000004	TCBREADY	DS.L	1	LINK FOR READY LIST
00000010	00000004	TCBNAME	DS.L	1	4-BYTE TASK NAME
00000014	00000004	TCBSESSN	DS.L	1	SESSION CODE
00000018	00000008	TCBMON	DS.L	2	MONITOR TASKNAME + SESSION CODE
00000020	00000004	TCBSEM	DS.L	1	LINK TO NEXT SEMAPHORE WAITER
00000024	00000001	TCBPR1	DS.B	1	CURRENT TASK PRIORITY
00000025	00000001	TCBPR2	DS.B	1	TASK LIMIT PRIORITY
00000026	00000001	TCBPR3	DS.B	1	PRIORITY USED TO ENTER TASK IN READY LIST
00000027	00000001	TCBIOCNT	DS.B	1	COUNT OF PENDING INPUTS
00000028	00000002	TCBATTR	DS.W	1	TASK ATTRIBUTES
00000029	00000029	TCBATTR1	EQU	TCBATTR+1	2ND BYTE OF ATTRIBUTES (INTERNAL FLAGS)
0000002A	00000002	TCBABORT	DS.W	1	ABORT CODE
0000002C	00000004	TCBSTATE	DS.L	1	CURRENT TASK STATE
0000002D	0000002D	TCBSTAT2	EQU	TCBSTATE+1	2ND BYTE OF STATE WORD
00000030	00000006	TCBTSTSM	DS.W	3	TST SEMAPHORE
00000036	00000004	TCBTST	DS.L	1	POINTER TO TASK SEGMENT TABLE
0000003A	00000006	TCBASQSM	DS.W	3	ASQ SEMAPHORE
00000040	00000004	TCBASQ	DS.L	1	POINTER TO ARQ
00000044	00000004	TCBCHAN	DS.L	1	LINK TO NEXT CHANNEL CONTROL BLOCK
00000048	00000004	TCBEVECT	DS.L	1	ADDRESS OF TASK EXCEPTION VECTOR
0000004C	00000004	TCBTVECT	DS.L	1	ADDRESS OF TASK TRAP VECTOR
00000050	00000004	TCBTACC	DS.L	1	ACCUMULATED TIME FOR THIS TASK (MSEC)
00000054	00000004	TCBTDLX	DS.L	1	TIME TO RESTART JOB AFTER DELAY
00000058	00000004	TCBDLYLK	DS.L	1	LINK TO NEXT TCB IN DELAY LIST
0000005C	00000002	TCBUPDO	DS.W	1	SAVE UPPER 1/2 OF DO ON TRAP 1'S
0000005E	00000002	TCBISRS	DS.W	1	ISR ERROR CODE - SAVE FOR WAKEUP
00000060	00000004	TCBAINT	DS.L	1	INTERVAL FOR PERIODIC ACTIVATION
00000064	00000004	TCBAASR	DS.L	1	ADDR OF ASR FOR PERIODIC ACTIVATION
00000068	00000002	TCBAOPT	DS.W	1	PERIODIC ACTIVATION OPTIONS
0000006A	00000002	TCBENTRY	DS.W	1	TASK INITIAL ENTRY POINT
0000006C	00000004	TCBUSER	DS.W	1	USER NUMBER ASSOCIATED WITH TASK
00000070	00000002	TCBSSP	DS.B	1	EXEC STACK DEPTH
00000072	00000001	TCBUTRP	DS.B	1	USER TRAP NUMBER - SET WHEN TRAP OCCURS
00000073	00000001				

0000000F	TSKASYST EQU	15	SYSTEM TASK
0000000E	TSKAMRES EQU	14	TASK IS MEMORY RESIDENT
0000000D	TSKACRIT EQU	13	TASK IS CRITICAL TO OS - CRASH SYSTEM IF ABORTED
0000000B	TSKARELO EQU	11	TASK IS RELOCATABLE (NO MMU) - CONVERT ENTRY
00000007	TSKIUSEM EQU	7	TASK HAS CREATED USER SEMAPHORE
00000006	TSKIEUXM EQU	6	TASK CONTROLLED BY EXCEPTION MONITOR
00000005	TSKIEIXMT EQU	5	TASK IS EXCEPTION MONITOR FOR ANOTHER TASK
00000004	TSKIEVCT EQU	4	TASK HAS OWN EXCEPTION VECTOR
00000003	TSKITVCT EQU	3	TASK HAS OWN TRAP VECTOR
00000002	TSKILAST EQU	2	TASK IS LAST TASK IN SESSION (SET ONLY BY TERM)
00000001	TSKLABRT EQU	1	TASK WAS ABORTED
00000000	TSKITVCT EQU	0	TASK HAS CLAIMED USER VECTOR

* タスク属性の定義

0000000F	TSKSDORM EQU	15	TASK IS DORMANT
0000000E	TSKSBLOC EQU	14	TASK IS BLOCKED
0000000D	TSKSSMWT EQU	13	TASK IS BLOCKED ON EXEC SEMAPHORE
0000000C	TSKSEVWT EQU	12	TASK IS WAITING FOR EVENT
0000000B	TSKSAKWT EQU	11	TASK IS WAITING FOR SERVICE REQ ACKNOWLEDGEMENT
0000000A	TSKSWTEM EQU	10	TASK IS WAITING FOR COMMAND FROM EXMON
00000009	TSKSSUSP EQU	9	TASK IS SUSPENDED
00000007	TSK2TRMP EQU	7	TASK HAS PENDING TERMINATION
00000006	TSK2RTEX EQU	6	TASK WILL RETURN TO EXEC
00000005	TSK2EVWK EQU	5	TASK IS HEADED FOR ASR
00000004	TSK2NRDY EQU	4	TASK IS ON READY LIST
00000003	TSK2WKWT EQU	3	TASK HAS PENDING WAKEUP
00000002	TSK2ACK2 EQU	2	TERM MESSAGE TO SERVER SENT WHILE ACK OUTSTANDING

* タスク状態の定義

00000001	TEMTRAP1 EQU	1	TRAP 1 IS MONITORED
00000010	TEMEUSER EQU	16	BUS ERROR IS MONITORED
00000011	TEMADDR EQU	17	ADDRESS ERROR IS MONITORED
00000012	TEMILLEG EQU	18	ILLEGAL INSTRUCTION IS MONITORED
00000013	TEMZDIV EQU	19	ZERO DIVIDE IS MONITORED
00000014	TEMCHK EQU	20	CHK INSTRUCTION IS MONITORED
00000015	TEMTRAPV EQU	21	TRAPV INSTRUCTION IS MONITORED
00000016	TEMPRIV EQU	22	PRIVILEGE VIOLATION IS MONITORED
00000017	TEML1010 EQU	23	LINE 1010 IS MONITORED
00000018	TEML1111 EQU	24	LINE 1111 IS MONITORED

*

MAX INSTRUCTION COUNT SUPPLIED
SINGLE INSTRUCTION TRACE REQUESTED (IF VCHG=0)
VALUE = TEST REQUESTED (IF VCHG=1)
VALUE CHANGE TEST REQUESTED

* タスク・マスクの定義

TEMCMONT EQU 27
TEMTRAC EQU 28
TEMVEQU EQU 28
TEMVCHG EQU 29

00000000 00000004
00000004 00000001
00000005 00000001
00000006 00000002
00000008 00000002
0000000A 00000002
0000000C 00000020
0000002G 00000020
0000004C 00000004
00000050 00000050

TST DS.L
TSTNSEGS DS.B
TSTCSEGS DS.B
TSTLMMU DS.W
TSTLATR DS.W
TSTSTAT DS.W
TSTMU DS.W
TSTATTR DS.W
TSTWISK DS.L
TSTLEN EQU

1
1
1
1
1
1
4*SEGMENTS
1
1
*

!TST DEBUG TOOL
NUMBER OF SEGMENTS
NUMBER OF SEGMENTS CURRENTLY EXISTING
INDEX TO LAST MMU LOAD LINE
INDEX TO LAST SEGMENT ATTRIBUTE LINE
MMU STATUS
MMU LOAD INFORMATION
SEGMENT ATTRIBUTE BLOCK
NAME OF TASK WITH WINDOW TO ADDRESS SPACE
LENGTH OF TASK SEG TABLE

* TST の構成

0000000F
0000000E
0000000D
0000000C
0000000B
0000000A

SEGAUSED EQU 15
SEGAROM EQU 14
SEGASHR EQU 13
SEGAGLBL EQU 12
SEGAMID EQU 11
SEGAPHOM EQU 10

SEGMENT ENTRY IN USE
SEGMENT IS READ ONLY
SEGMENT IS SHARED WITHIN SESSION
SEGMENT IS GLOBAL
SEGMENT IS MEMORY MAPPED TO SPACE
SEGMENT IS PHYSICAL ROM

* セグメント属性ビットの定義

00000000
00000002
00000004
00000006

TSTLB EQU 0
TSTLE EQU 2
TSTPO EQU 4
TSTCTL EQU 6

LOGICAL BEGIN
LOGICAL END
PHYSICAL OFFSET
CONTROL (NULL BYTE + CONTROL BYTE)

* MMU 制御レジスタ値

図 11.16 TST: タスク・セグメント・テーブルの例

タイプ1セマフォ例

タスク A	タスク B	タスク C	セマフォ BUFRDY			セマフォ MSGSNT		
			初 カ ウ ン ト	期 信 号 カ ウ ン ト	FIFO	初 期 信 号 カ ウ ン ト	期 信 号 カ ウ ン ト	FIFO
	ATSEM (BUFRDY)		5	5	空である	0	0	空である
CRSEM (BUFRDY) CRSEM (MSGSNT) WTSEM (MSGCNT)							-1	タスク A
	ATSEM (MSGCNT) WTSEM (BUFRDY) (メッセージをバッファ に入れる)			4				
(メッセージを処理する) SGSEM (BUFRDY) WTSEM (MSGCNT)				5			0	空となる
		ATSEM (BUFRDY) ATSEM (MSGSNT) WTSEM (BUFRDY) (メッセージをバッファ に入れる)		4			-1	タスク A
		SGSEM (MSGCNT) WTSEM (BUFRDY) (メッセージをバッファ に入れる)		3			0	
(メッセージを処理する)		SGSEM (MSGSNT)					1	空となる

時間 →

タイプ2セマフォ例

タスク A	タスク B	タスク C	R1SEM		
			初 カ ウ ン ト	期 信 号 カ ウ ン ト	FIFO
				1 0	空である
ATSEM	ATSEM WTSEM				

時間 →

WTSEM	ATSEM WTSEM	-1	タスクAがセマフォ・キューに登録される
(資源を使用する) SGSEM		-2	セマフォ・キューにさらにタスクCが追加される
(資源を使用する) SGSEM		-1	タスクAがセマフォ・キューから削除され、実行可リストに移される
	(資源を使用する) SGSEM	0	タスクCがセマフォ・キューから削除され、実行可リストに移される
		1	

タイプ3セマフォ例

タスクA	タスクB	タスクC	セマフォR2ASEM			セマフォR2BSEM		
			初 カ ウ ン ト	期 信 カ ウ ン ト	号	FIFO	初 カ ウ ン ト	期 信 カ ウ ン ト
ATSEM(A) CRSEM(B) WTSEM(A)	ATSEM(A) CRSEM(B) WTSEM(A)		0			空である	0	
CRSEM(A) (処理を行う) SGSEM(A) DESEM(A)			-1			タスクB	0	0
DESEM(A) (処理を行う) SGSEM(B) DESEM(B)			0			空となる		1
	ATSEM(B) WTSEM(B) DESEM(B) (処理を行う)							0

時間 →

この時点でR2ASEMは削除される

この時点でR2BSEMは削除される

図 11・17 セマフォの例

11 開発ソフトウェア/リアルタイム・モニタ

LOC	OBJ	SOURCE	< Assembler for MC68k on CP/M >
		<pre> : : : MONITOR PROGRAM : : </pre>	
		<pre> =000F ACIAC EQU 0F8001H ; ACIA CONTROL REGISTER 8001 =000F ACIAD EQU 0F8003H ; ACIA DATA REGISTER 8003 =0000 RECREADY EQU 0 ; RECEIVE READY FLAG 0000 =0000 SENDREADY EQU 1 ; SEND READY FLAG 0001 =0000 SENDOVER EQU 3 ; SEND OVER FLAG 0003 =0000 ESCAPE EQU 82H ; ESCAPE CHARACTER WITH PARITY 0082 =0002 ENDOBRAM EQU 20000H ; END OF RAM 0000 =0001 SSPBOT EQU 10000H ; SYSTEM STACK BOTTOM 0000 =0000 CONTAD EQU 10000H-4 ; CONTINUATION ADDRESS FFFC =0000 STATUS EQU 10000H-6 ; USER STATUS SAVE AREA ADDRESS FFFA =0000 UREGSV EQU 10000H-70 ; TOP OF STACK WHEN USER REG'S ARE SAVED FFFA =0000 CLEAR EQU 0 ; ZERO NULL NIL NASHI 0000 =0000 BS EQU 7FH ; BACK SPACE 007F =0000 CAN EQU 18H ; CANCEL 0018 =0000 CR EQU 0DH ; RETURN 0000 =0000 LF EQU 0AH ; LINE FEED (NEGLECT) 000A =0000 DUMMYVAL EQU 0 ; DUMMY VALUE 0000 </pre>	
0000	0000	<pre> =0000 ORG 0H 0000 0000 0000 DEFL SSPBOT ; INITIAL SSP 0001 0002 0000 DEFL MONITSTART ; INITIAL PC 0004 0005 0000 0006 0500 0000 0008 0000 DEFL ERROR ; BUS ERROR 000A 0300 0000 000C 0000 DEFL ADDRERR ; ADDRESS ERROR 000E 0278 0000 0010 0000 DEFL LLLDGL ; ILLDGL INSTRUCTION 0012 0322 0000 0014 0000 DEFL ZERO ; ZERO DIVIDE 0016 0312 0000 0018 0000 DEFL ERROR ; CHK INSTRUCTION 001A 0300 0000 001C 0000 DEFL ERROR ; TRAPV INSTRUCTION 001E 0300 0000 0020 0000 DEFL PRIVERR ; PRIVLPGE VIOLATION 0022 0332 0000 0024 0000 DEFL TRACE ; TRACE 0026 0356 0000 0028 0000 DEFL ERROR ; LINE 1010 EMULATOR 002A 0300 0000 002C 0000 DEFL ERROR ; LINE 1111 EMULATOR 002E 0300 0000 0030 0000 0030 ORG 60H 0000 0060 0062 0300 DEFL ERROR ; SPURIOUS INTERRUPT 0064 0000 DEFL ERROR ; AUTOVECTOR 1 0066 0300 </pre>	

12. MC 68000 の 将来の発展方向

マイクロプロセッサは、チップ寸法の制約、プロセス技術の制約、性能の要求など多くの制約の中でおのものの機構を取り入れ、よりいっそう強化する方向に向かっております。アーキテクチャやソフトウェア面も、より強力なモニタが作成され、その管理下のもとにシステムが運用されるようになります。MOS 技術の方向、ソフトウェアやハードウェア・コストの傾向と 68000 の発展方向を説明しています。

12.1 MOS技術の方向

マイクロプロセッサの生産における MOS 技術の進歩を図 12.1 に示します。1973 年の 8 ビット MPU スタート時の NMOS 技術と現在の 32 ビット MPU の HMOS 技術との間に容量/チップを例にしても約 10 倍以上と数段の躍進がなされたことがわかりいただけるでしょう。

図 12.2 は、年々の技術躍進をなしたバックグラウンド資料として、ダイ・サイズの傾向を載せておきました。ダイ・サイズにおいても 8 ビットときと比較して 3 倍以上の技術進歩がなされています。

年 ↓	項目 種類	ゲート・レングス	エリア・ファクタ	ゲート・ディレイ (TIME t_D)	容量/チップ
	標準 NMOS	6. MICRONS	1	10 ns	50 k
	HMOS I	3.5 MICRONS	.48	4~5 ns	100 k
	HMOS II	2.5 MICRONS	.26	2~3 ns	250 k
	HMOS III	1.7 MICRONS	.15	1 ns	650 k

図 12.1 MOSテクノロジーとパフォーマンスの発展

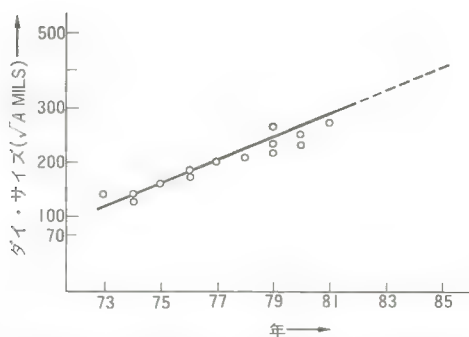


図 12.2 マイクロプロセッサのダイ・サイズと傾向

12・2 ソフトとハード・コストの方向

ソフトウェア・コストとハードウェア・コストは、図 12・3 に示すように反比例の関係になりつつあり、年々ハードウェア・コストは減少し、逆にソフトウェア・コストは増加しています。このことは、プロセス技術と生産設備技術の著しい躍進があげられます。この傾向は、今後なおいっそう増幅されてゆくと思われます。図 12・4 はシステム・コストの傾向を示し、ハードウェアとソフトウェアの分岐が 1985 年ごろに完全に逆転し、ソフトウェア・コストがハードウェア・コストを大幅に上回ってくることです。

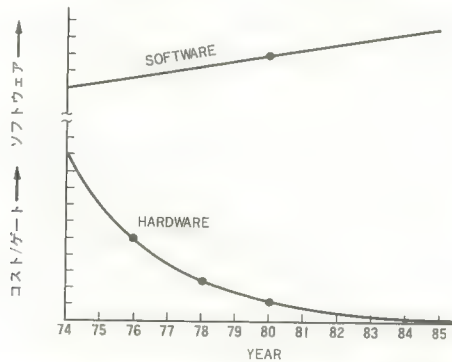


図 12・3 ソフトウェアとハードウェア・コストの推移

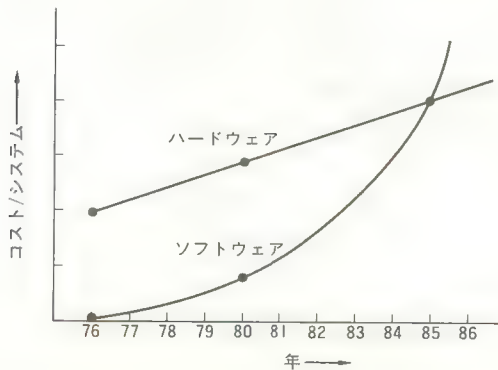


図 12・4 ハードウェアとソフトウェアのシステム・コスト

12.3 MC 68000 の発展方向

モトローラのマイクロプロセッサの**設計サイクル**を見ますと、図 12.5 に示すように 5 ～ 6 年をサイクルに展開されています。1973 から 1979 年までは 8 ビット系の確立に、1979 から 1985 年までは 16 ビットの確立に、32 ビットの確立には 1987 年ごろまで費やされるでしょう。

仮想マシン (VM) プロセッサ MC 68010 は、MC 68000 では実メモリ空間に命令がないときにホルトが起きても、その命令をアボードしてリランする機能がなかったので、アボード機能を追加し、例外ベクタ・テーブルを再配置することによって仮想メモリを実現しています。また、例外ベクタ・テーブルを動的に変更できるようになったことで、仮想マシンを複数個持たせることができるようになりました。

MC 68008 は縮小バス・プロセッサであり、価格や開発費の低減を目的としており、アドレス・バスが 20 ビットでデータ・バスが 8 ビットです。

32 ビット・マイクロプロセッサ **MC 68020** は、メモリ・アドレスが 16 M バイトから 4 G バイトまでアクセス可能であり、コプロセッサ MC 68881 に接続できるアーキテクチャを持ちます。使用トランジスタは 15 万個で、HMOS III のプロセス技術で作られています。

浮動小数点コ・プロセッサ MC 68881 は、32 ビット・プロセッサ MC 68020 用結合プロセッサであり、MC 68000 および MC 68010 のペリフェラルとしても使用できるようになっています。使用トランジスタは 12 万個で、HMOS III のプロセス技術で作られます。

12 68000 の将来の発展方向

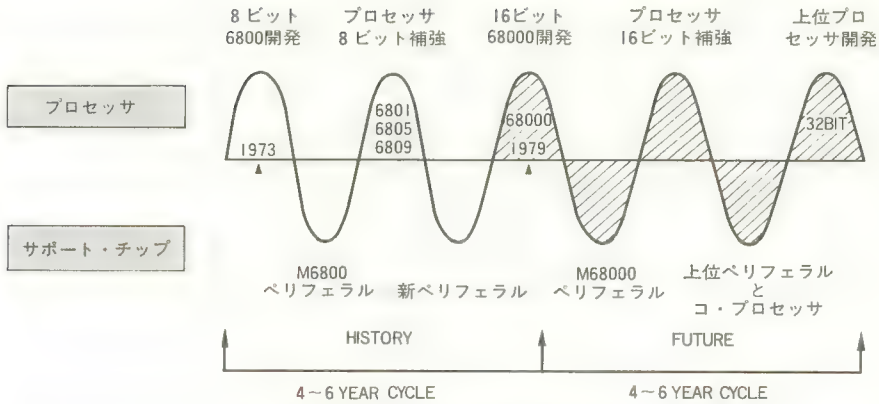


図 12・5 モトローラ生産開発プラン

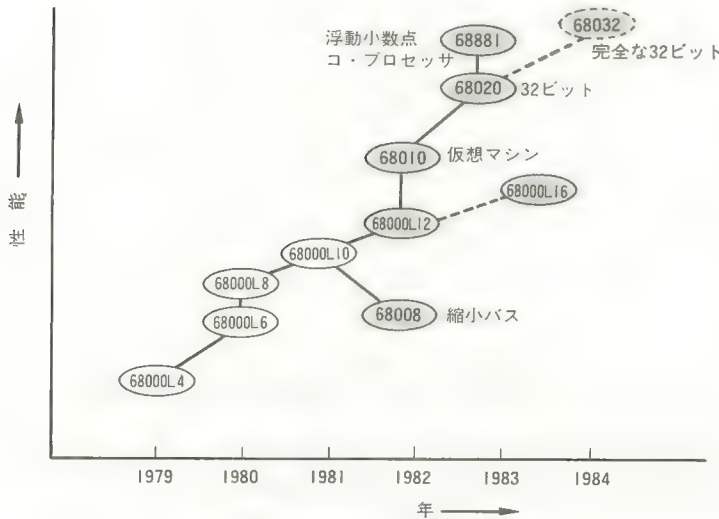


図 12・6 MC 68000 ファミリの開発予定

1983 年発売予定——周辺サポート・チップ——

MC 68200	MCU: マイクロ・コンピュータ・ユニット
MC 68430	DMAI: ダイレクト・メモリ・アクセス・インタフェース
MC 68440	DOMA: デュアル・チャネル・ダイレクト・メモリ・アクセス
MC 68450	DMAC: ダイレクト・メモリ・アクセス・コントローラ
MC 68452	BAM: バス・アービトレーション・モジュール
MC 68454	IMDC: インテリジェント・マルチディスク・コントローラ
MC 68459	DPLL: ディスク・フェーズ・ロック・ループ
MC 68562	DUSCC: デュアル・ユニバーサル・シンクロナス・コミュニケーション・コントローラ
MC 68564	SIO: シリアル I/O
MC 68590	LANCE: ローカル・エリア・ネットワーク・コントローラ
MC 68901	MFP: マルチ・ファンクション・ペリフェラル

付録 1. レジスタ転送言語定義 (RTL: Register Transfer Language)

以下にレジスタ転送言語定義で命令セットの詳細な動作説明をします。

オペランド:

An: アドレス・レジスタ

Dn: データレジスタ

Rn: データ・レジスタまたはアドレス・レジスタのいずれか

PC: プログラム・カウンタ

SR: ステータス・レジスタ

CCR: コンディション・コード (SRの下位バイト)

SSP: スーパーバイザ・スタック・ポインタ

USP: ユーザ・スタック・ポインタ

SP: アクティブ・スタック・ポインタ (A7) と同じ

X: 拡張オペランド (コンディション・コードからの)

Z: ゼロ・コンディション・コード

V: オーバフロー・コンディション・コード

Immediate Data: イミディエート・データ・オペランド

d: アドレス変位 (displacement)

Source: ソース位置の指定

Destination: ディスティネーション位置の指定

Vector: エクセプション・ベクタの位置

サブフィールドと修飾記号 (qualifiers)

<ビット>OF<オペランド> オペランドのある一つのビットを選ぶ。

<オペランド>[<ビット番号>:<ビット番号>] オペランドのサブフィールドを選ぶ。

(<オペランド>) 参照されたオペランドの中身

<オペランド>₁₀ オペランドはBCD (2進法10進) であり、オペレーションは10進で実行される。

<オペランド>@<モード> オペランドで指定したアドレス・レジスタがオペランド・データのメモリ位置を指示することを表すレジスタ間接オペレータ・オプションのモード修飾記号は、-, +, (d), および (d, ix)。

オペレーション: オペレーションは、2オペランド操作、単一オペランド操作およびその他に分けられる。

2オペランド操作: これらの命令は<オペランド><オペレーション><オペランド>で表され、ここで
の<オペレーション>は次のいずれかである。

→ 左のオペランドが右のオペランドで指定される位置に移される。

↔ 二つのオペランド内容が交換される。

- + 両オペランドが加算される。
- 右側のオペランドが左側のオペランドから減算される。
- * 左右のオペランドが乗算される。
- / 左のオペランドが右のオペランドで除算される。
- ^ 左右オペランドが論理 AND される。
- v 左右オペランドが論理 OR される。
- ⊕ 左右オペランドが論理 EOR される。
- < 関係テスト、左のオペランドが右のオペランドより小なら真。
- > 関係テスト、左のオペランドが右のオペランドより大なら真。
- ≡ 関係テスト、左のオペランドが右のオペランドに等しくなければ真。

shifted by } 左のオペランドが、右のオペランドで規定される位置数だけシフト、またはローテートされ
rotated by } る。

単一オペランド：

～<オペランド> オペランドの論理的補数をとる。

<オペランド>Sign-extended オペランドが符号拡張される。上位半分の全ビットが下位半分の最上位ビットと等しくされる。

<オペランド>tested オペランドが 0 と比較され、その結果がコンディション・コードのセットに使用される。

その他：

TRAP PC→SSP@-;SR→SSP@-;(ヘクタ)→PCS と等価。

STOP ストップ状態に入り、割込みを待つ。

If<コンディション>then<オペレーション>else<オペレーション>

そのコンディションでテストされる。もし真なら then の次の操作が実行される。もしコンディションが偽でオプションの else 部があれば else の操作が実行される。もしコンディションが偽でオプションの else 部がなければ、命令は操作されない。

付録 2. MC 68000 アセンブラ命令一覧表

(1)

分類	命令	オペランド・サイズ			オペランド	モード	操作	作	フラグ					備考	
		B	W	L					X	N	Z	V	C		
データ転送命令	MOVE	●	●	●	<ea>, <ea>	→			—	*	*	0	0	{ des→DA soc→AL (Byte→An モード不可)	
	MOVEA		●	●	<ea>, An	AL			—	—	—	—	データ範囲 (—\$80~\$7F)		
	MOVEP		●	●	Dx, d(Ay)	—			—	—	—	—			(An)@+ モード可
			●	●	d(Ay), Dx	—	(soc)→des		—	—	—	—			
	MOVEQ			●	#<data>, Dn	—		Rx←→Ry	—	—	—	—		EXG, Dn, An	
	MOVEM		●	●	<reg-list>, <ea>	CA		{soc}→An	—	—	—	—	Dn←→An に限り EXG, Dn, An		
			●	●	<ea>, <reg-list>	C			—	—	—	—		Dn←→An に限り EXG, Dn, An	
	EXG			●	Rx, Ry	—			—	—	—	—	Dn←→An に限り EXG, Dn, An		
	LEA			●	<ea>, An	—	C			—	—	—		Dn←→An に限り EXG, Dn, An	
	SWAP			●	Dn	—		REG[31:16]←→REG[15:0]		—	—	—	Dn←→An に限り EXG, Dn, An		
PEA			●	<ea>	C		des→SP@—		—	—	—	Dn←→An に限り EXG, Dn, An			
LINK		—	—	An, #<disp>	—		An→SP@—, SP→An, SP+d→SP		—	—	—		Dn←→An に限り EXG, Dn, An		
UNLK		—	—	An	—		An→SP, SP@+→An		—	—	—	Dn←→An に限り EXG, Dn, An			
演算命令	ADD	●	●	●	<ea>, Dn	AL				*	*		*	*	Byte→An モード不可
		●	●	●	Dn, <ea>	AM	(des)+(soc)→des		*	*	*	*	Byte→An モード不可		
	ADDA		●	●	<ea>, An	AL				—	—	—		Byte→An モード不可	
	ADDI	●	●	●	#<data>, <ea>	DA	(des)+ID→des		*	*	*	*	Byte→An モード不可		
	ADDQ	●	●	●	#<data>, <ea>	A	(des)+ID(0~7)→des		*	*	*	*		Byte→An モード不可	
		●	●	●	Dy, Dx	—				*	*	*	*		Byte→An モード不可
		●	●	●	-(Ay), -(Ax)	—	(des)+(soc)+X→des		*	*	*	*	Byte→An モード不可		
	SUB		●	●	<ea>, Dn	AL				*	*	*		*	Byte→An モード不可
		●	●	●	Dn, <ea>	AM	(des)-(soc)→des		*	*	*	*	Byte→An モード不可		
	SUBA		●	●	<ea>, An	AL				—	—	—		—	Byte→An モード不可


付

録

分 類	命 令	オペランド・サイズ			オ ペ ラ ン ド	モ ー ド	操 作	フ ラ グ					備 考	
		B	W	L				X	N	Z	V	C		
演 算	SUBI	●	●	●	#<data>, <ea>	DA	(des) - ID → des	*	*	*	*	*	*	Byte → An モード不可
	SUBQ	●	●	●	#<data>, <ea>	A	(des) - ID(0~7) → des	*	*	*	*	*	*	
	SUBX	●	●	●	Dy, Dx	-	(des) - (soc) - X → des	*	*	*	*	*	*	
		●	●	●	-(Ay), -(Ax)	-		*	*	*	*	*	*	
	MULS			●		D	(des) * (soc) → des	-	*	*	0	0	符号付き	
	MULU			●		D		-	*	*	0	0	符号なし	
	DIVS			●	<ea>, Dn	D		-	*	*	*	0	符号付き	
	DIVU			●		D	(des) / (soc) → des	-	*	*	*	0	符号なし	
	CLR	●	●	●		DA	0 → des	-	0	1	0	0		
	NEG	●	●	●		DA	0 - (des) → des	*	*	*	*	*	*	
命 令	NEGX	●	●	●	<ea>	DA	0 - (des) - X → des	*	*	*	*	*	*	
	TAS	●				DA	(des) tested → CC	-	*	*	0	0		
						DA	1 → bit 7 of des	-	*	*	0	0		
	TST	●	●	●		DA	(des) tested → CC	-	*	*	0	0	データ 0 と比較	
	EXT		●	●	Dn	-	(des) 符号拡張 → des	-	*	*	0	0		
	CMP	●	●	●	<ea>, Dn	AL		-	*	*	*	*	Byte → An モード不可	
	CMPL		●	●	<ea>, An	AL	(des) - (soc)	-	*	*	*	*		
	CMPL	●	●	●	(Ay) +, (Ax) +	-		-	*	*	*	*		
	CMPI	●	●	●	#<data>, <ea>	DA		-	*	*	*	*		
	AND	●	●	●	<ea>, Dn	D	(des) ^ (soc) → des	-	*	*	0	0		
論 理 命 令	ANDI	●	●	●	Dn, <ea>	AM		-	*	*	0	0		
	OR	●	●	●	#<data>, <ea>	DA	(des) / ID → des	-	*	*	0	0	SR モード可	
		●	●	●	<ea>, Dn	D	(des) v (soc) → des	-	*	*	0	0		
		●	●	●	Dn, <ea>	AM		-	*	*	0	0		
ORI	●	●	●	#<data>, <ea>	DA	(des) v ID → des	-	*	*	0	0	SR モード可		

(2)

分類	命令	オペランド・サイズ			オペランド	モード	操 作	フラグ				備 考	
		B	W	L				X	N	Z	V		C
論理命令	EOR	●	●	●	Dn, <ea>	DA	$(des) \oplus (src) \rightarrow des$	—	●	●	0	0	SRモード可
	EORI	●	●	●	#<data>, <ea>	DA	$(des) \oplus ID \rightarrow des$	—	●	●	0	0	
	NOT	●	●	●	<ea>	DA	$\sim (des) \rightarrow des$	—	●	●	0	0	
シフト命令	ASL	●	●	●	Dx, Dy	—		●	●	●	●	●	
		●	●	●	#<data>, Dy	—		●	●	●	●	●	
		●	●	●	<ea>	AM		●	●	●	●	●	
シフト命令	ASR	●	●	●	Dx, Dy	—		●	●	●	●	●	
		●	●	●	#<data>, Dy	—		●	●	●	●	●	
		●	●	●	<ea>	AM		●	●	●	●	●	
シフト命令	LSL	●	●	●	Dx, Dy	—		●	●	0	●	●	
		●	●	●	#<data>, Dy	—		●	●	0	●	●	
		●	●	●	<ea>	AM		●	●	0	●	●	
シフト命令	LSR	●	●	●	Dx, Dy	—		●	●	0	●	●	メモリ・シフトは1ビットのみシフト
		●	●	●	#<data>, Dy	—		●	●	0	●	●	
		●	●	●	<ea>	AM		●	●	0	●	●	
ローテーション命令	ROL	●	●	●	Dx, Dy	—		—	●	●	0	●	
		●	●	●	#<data>, Dy	—		—	●	●	0	●	
		●	●	●	<ea>	AM		—	●	●	0	●	
ローテーション命令	ROR	●	●	●	Dx, Dy	—		—	●	●	0	●	
		●	●	●	#<data>, Dy	—		—	●	●	0	●	
		●	●	●	<ea>	AM		—	●	●	0	●	
ローテーション命令	RJXL	●	●	●	Dx, Dy	—		—	●	●	0	●	
		●	●	●	#<data>, Dy	—		—	●	●	0	●	
		●	●	●	<ea>	AM		—	●	●	0	●	

分類	命令	オペランド・サイズ			オペランド	モード	操 作	フ ラ グ					備 考
		B	W	L				X	N	Z	V	C	
ロジック命令	ROXR	●	●	●	Dx, Dy	—		*	*	*	0	*	メモリ・ローテーションは1ビットのみシフト
		●	●	●	#<data>, Dy	—		*	*	*	0	*	
		●	●	●	<ea>	AM		*	*	*	0	*	
ビット操作命令	BTST	●	●	●	Dn, <ea>	D	~(指定ビット)→Z	—	*	*	—	—	IDモード不可↓
		●	●	●	#<data>, <ea>	D		—	*	*	—	—	
	BSET	●	●	●	Dn, <ea>	DA	~(指定ビット)→Z	—	*	*	—	—	2ワード目にビット番号
		●	●	●	#<data>, <ea>	DA	1→(指定ビット)	—	*	*	—	—	
	BCLR	●	●	●	Dn, <ea>	DA	~(指定ビット)→Z	—	*	*	—	—	2ワード目にビット番号
		●	●	●	#<data>, <ea>	DA	0→(指定ビット)	—	*	*	—	—	
	BCHG	●	●	●	Dn, <ea>	DA	~(指定ビット)→Z	—	*	*	—	—	2ワード目にビット番号
		●	●	●	#<data>, <ea>	DA	~(指定ビット)→指定ビット	—	*	*	—	—	
BCD命令	ABCD	●	●	●	Dy, Dx	—	(des) ₁₀ + (soc) ₁₀ + X → des	*	u	*	u	*	2ワード目にビット番号
		●	●	●	-(Ay), -(Ax)	—		*	u	*	u	*	
	SBCD	●	●	●	Dy, Dx	—	(des) ₁₀ - (soc) ₁₀ - X → des	*	u	*	u	*	
		●	●	●	-(Ay), -(Ax)	—	*	u	*	u	*		
フロッピー命令	NBCD	●	●	●	<ea>	DA	0 - (des) ₁₀ - X → des	*	u	*	u	*	PC@ (d) 符号付き 8 or 16ビット
	Bcc	●	●	●	<label>	—	if cc then PC+d→PC	—	—	—	—	モードのみ可 符号付き 16ビット	
	DBcc	●	●	●	Dn, <label>	—	if ~cc then Dn-1→Dn if Dn≠-1 then PC+d→PC	—	—	—	—		モードのみ可 符号付き 8 or 16ビット
	Scc	●	●	●	<ea>	DA	if cc then I's→des else 0's→des	—	—	—	—		
	BRA	●	●	●	<label>	—	PC+d→PC	—	—	—	—		
	BSR	●	●	●	<label>	—	PC→SP@-, PC+d→PC	—	—	—	—		

(3)

分類	命令	オペランド・サイズ			オペランド	モード	操 作	フラグ					備 考
		B	W	L				X	N	Z	V	C	
プロセッサ命令	JMP	—	—	—	<ea>		C (des)→PC	—	—	—	—	—	
	JSR	—	—	—	<ea>		C PC→SP@-, (des)→PC	—	—	—	—	—	
	RTR	—	—	—		—	SP@+→CC, SP@+→PC	*	*	*	*	*	
	RTS	—	—	—		—	SP@+→PC	—	—	—	—	—	
システム・コントロール	RESET	—	—	—			外部デバイスのリセット	—	—	—	—	—	
	RTE	—	—	—		—	SP@+→SR, SP@+→PC	*	*	*	*	*	
	STOP	—	—	—	#xxx	—	ID→SR, stop	*	*	*	*	*	
	MOVE			●	USP, An	—	(USP)→An	—	—	—	—	—	
				●	An, USP	—	An→USP	—	—	—	—	—	
	ANDI		●		<ea>, SR	D	(soc)→SR	*	*	*	*	*	
	ORI		●			—	(SR)∧ID→SR	—	*	*	*	0 0	
	EORI		●		#<data>, SR	—	(SR)∨ID→SR	—	*	*	*	0 0	
						—	(SR)⊕ID→SR	—	*	*	*	0 0	
	TRAP	—	—	—	#<vector>	—	PC→SSP@-, SR→SSP@- vector→PC	—	—	—	—	—	
TRAPV	—	—	—		—	if V then TRAP	—	—	—	—	—		
命令	CHK		●		<ea>, Dn	D	if Dn<0 or Dn>(soc) then trap	—	*	u	u	u	
	ANDI	●					(CCR)∧ID→CCR	—	*	*	*	0 0	
	ORI	●			#<data>, CCR		(CCR)∨ID→CCR	—	*	*	*	0 0	
	EORI	●					(CCR)⊕ID→CCR	—	*	*	*	0 0	
	MOVE		●		<ea>, CCR	D	(soc)→CCR	*	*	*	*	*	
		●		SR, <ea>	DA	SR→(des)	—	—	—	—	—	—	
その他	NOP	—	—	—				—	—	—	—	—	何の処理もしない

特権命令

トフリップ発生命令

ステータス・レジスタ変更命令

(4)

実効アドレス・モード <ea>

(X:指定可)

実効アドレス・モード	F A	アドレス・モード							<ea>	ディスプレイメント
		MOD	REG	D	M	C	A	DA/CA/MA/L		
Dn	000	番号	X				X	X	Dn	—
An	001	番号	X				X		An	—
An@	010	番号	X	X	X	X	X	X	(An)	—
An@+	011	番号	X	X	X	X	X	X	(An)+	—
An@-	100	番号	X	X	X	X	X	X	-(An)	—
An@(d)	101	番号	X	X	X	X	X	X	d(An)	—
An@(d,ix)	110	番号	X	X	X	X	X	X	d(An,Rn,W) d(An,Rn,L)	符号付き 16ビット 符号付き 8ビット
xxx.W	111 000		X	X	X	X	X	X	xxx	—
xxx.L	111 001		X	X	X	X	X	X	xxxxxx	—
PC@(d)	111 010	X	X	X	X	X	X	X	*+<ab,sym> *-<ab,sym> <rel,sym> <ab,sym>	符号付き 16ビット
PC@(d,ix)	111 011	X	X	X	X	X	X	X	<rel,sym>(Rn,W) <rel,sym>(Rn,L)	符号付き 8ビット
#xxx	111 100	X	X	X	X	X	X	X	#xxx	—
SR	111 100								SR,CCR	特殊モード

付

プログラム・コントロール命令での
コンディション(cc)

ニモ ニック	コンディション	エン コード	テスト条件
T	true	0000	1
F	false	0001	0
HI	high	0010	$\bar{C} \cdot \bar{Z}$
LS	low or same	0011	$C + Z$
CC	carry clear	0100	C
CS	carry set	0101	C
NE	not equal	0110	\bar{Z}
EQ	equal	0111	Z
VC	overflow clear	1000	\bar{V}
VS	overflow set	1001	V
PL	plus	1010	\bar{N}
MI	minus	1011	N
GE	greater or equal	1100	$N \cdot V + \bar{N} \cdot \bar{V}$
LT	less then	1101	$N \cdot \bar{V} + \bar{N} \cdot V$
GT	greater than	1110	$N \cdot \bar{V} \cdot \bar{Z} + \bar{N} \cdot \bar{V} \cdot Z$
LE	less or equal	1111	$Z + N \cdot \bar{V} + \bar{N} \cdot V$

符号ビット
考慮さ
れる

録

付録 3. オペレーション・コード表

オペレーション・コード・マップ

Bit 15 thru 12	オ ペ レ シ ョ ン
0000	Bit Manipulation/MOVP/Immediate
0001	Move Byte
0010	Move Loog
0011	Move Word
0100	Miscellaneous
0101	ADDO/SUBO/ScC/DBcc
0110	Bcc,BSR
0111	MOVEQ
1000	OR/DIV/SCD
1001	SUB/SUBX
1010	(Unassigned)
1011	CMP/EOR
1100	AND/MUL/ABCD/EXG
1101	ADD/ADDX
1110	Shift/Rotate
1111	(Unassigned)

Dynamic Bit

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 Register 1 Type Effective Address

Static Bit

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 1 0 0 0 Type Effective Address
Bit Type Codes: TST-00, CHG-01, CLR-10, SET-11

MOVEP

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 Register Op Mode 0 0 1 Register
Op Mode: Word to Reg-100, Long to Reg-101, Word to Mem-110, Long to Mem-111

OR Immediate

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 Size Effective Address

AND Immediate

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 1 0 Size Effective Address

SUB Immediate

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 1 0 Size Effective Address

ADD Immediate

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 1 0 Size Effective Address

EOR Immediate

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 1 0 1 0 Size Effective Address

CMP Immediate

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 1 1 0 0 Size Effective Address

MOVE Byte

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 1 Destination Register Mode Source Register

付

録

錄

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	1	0	Destination Register				Mode				Source Register			

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	0	1	1					Destination Register Mode		Source Register Mode					

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	0	0	0	Size	Effective Address					

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	0	0	1	1	Effective Address					

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	1	0	Size	Effective	Address					

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	1	0	0	Size	Effective Address						

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	1	0	0	1	1						
Effective Address															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	1	1	0	Size	Effective Address						

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	1	1	0	1	1						Effective Address

0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	Effective Address
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	0	0	0	1						
Effective Address															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	0	0	0	1	0	0	0	0	0	Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	0	0	1	Sz	Effective Address					

Sz: Long-1, Word-0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	0	0	1	0	0	0	0	0	0	Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	0	0	1	1	0	0	0	0	0	Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	1	0	Size		Effective Address					

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	1	0	1	1						Effective Address

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	0	0	1	Sz	Effective Address					

Sz: Long-1, Word-0

TRAP

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	0	0	Vector			

LINK

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	0	1	0	Register		

UNLK

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	0	1	1	Register		

MOVE to USP

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	0	0	Register		

MOVE from USP

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	0	1	Register		

RESET

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	0	0	0

NOP

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	0	0	1

STOP

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	0	1	0

RTE

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	0	1	1

RTS

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	0	1	0	1	0

TRAPV

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	1	0	0	1	1	0	1	1	0

RTR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	0	1	1	1	1

JSR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	1	0	1	0	Effective Address			

JMP

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	1	0	1	1	Effective Address			

CHK

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	Register				1	1	0	Effective Address				

LEA

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	Register				1	1	1	Effective Address				

ADDQ

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	Data				0	Size		Effective Address				

SUBQ

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	Data				1	Size		Effective Address				

付

罫

AND

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1	1	0	0	Register	Op mode	Effective Address								

Op Mode

.B .W .L
 000 001 010 Dn·EA→Dn
 100 101 110 EA·Dn→EA

MULU

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	Register		0	1	1	Effective Address						

MULS

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	Register		1	1	1	Effective Address						

ABCD

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	Destination Register	1	0	0	0	R/M	Source Register					

R/M (register/memory): register, register-0,
memory, memory-1

EXGD

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	Data Register	1	0	1	0	0	0	0	0	0	0	Register

EXGA

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	Address Register	1	0	1	0	0	1	0	0	1	Address Register	

EXGM

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	Data Register	1	1	0	0	0	1	Address Register				

ADD

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	1	Register	Op Mode	Effective Address									

Op Mode

.B .W .L
 000 001 010 Dn+EA→Dn
 100 101 110 EA+Dn→EA
 011 111 An+EA→An

ADDX

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	1	Destination Register	1	Size	0	0	R/M	Source Register					

R/M (register/memory): register, register-0,
memory, memory-1

Data Register Shifts

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	Count/ Register	d	Size	l/r	Type	Register						

Memory Shifts

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	Type	d	1	1	Effective Address							

Shift Type Codes: AS-00, LS-01, ROX-10, RO-11
 d (direction): Right-00, Left-1
 t/r (count source): Immediate Count-0,
Register Count-1

付録 4. 電気的仕様とタイミング図——MC 68000 L 4・L 6・L 8・L 12

4・1 電気特性

絶対最大定格

定 格	記 号	定 格 値	単 位
供給電圧	V_{CC}	$-0.3 \sim +7.0$	V
入力電圧	V_{in}	$-0.3 \sim +7.0$	V
動作温度	T_A	$0 \sim 70$	$^{\circ}\text{C}$
保存温度	T_{Stg}	$-55 \sim +150$	$^{\circ}\text{C}$

温度特性

特 性	記 号	特 性 値	単 位
熱抵抗 セラミック・パッケージ	θ_{JA}	30	$^{\circ}\text{C}/\text{W}$

このデバイスは、各入力に対する静電気あるいは高電圧に対する保護回路を備えています。応用上あらかじめハイ・インピーダンスの入力回路に、絶対最大定格を超えるような電圧がかからないように注意する必要があります。

未使用の入力ピンは、常に V_{SS} または V_{CC} に接続しなければなりません。

電力条件

チップの接合点の平均温度 $T_J(^{\circ}\text{C})$ は、次式で計算できます。

$$T_J = T_A + (P_D \cdot \theta_{JA})$$

ここで

T_A ≡ 周囲温度 ($^{\circ}\text{C}$)

θ_{JA} ≡ パッケージの熱抵抗, ジャンクション-周囲間 ($^{\circ}\text{C}/\text{W}$)

P_D ≡ $P_{INT} + P_{I/O}$

P_{INT} ≡ $I_{CC} \times V_{CC}$ (W) —— チップの内部消費電力

$P_{I/O}$ 入/出力端子での I/O 消費電力 (W) —— ユーザの使用方法による。

一般の使用条件では、 $P_{I/O} \ll P_{INT}$ のため無視できます。

P_{PORT} が無視できる場合の P_D と T_J の近代的な関係は、

(1)

$$P_D = K \div (T_J + 273^\circ\text{C})$$

(2)

となります。(1), (2)式からKを求めると,

$$K = P_D \cdot (T_A + 273^\circ\text{C}) + \theta_{JA} \cdot P_D^2$$

(3)

ここでKは既知の T_A における平衡状態の P_D を測定すれば, (3)式によって決定できる定数です。このKの値を(1), (2)式に代入し, 演算をすることによって, 任意の T_A における P_D と T_J を計算することができます。

DC電気的特性 ($V_{CC} = 5.0\text{ Vdc} \pm 5\%$, $V_{SS} = 0\text{ Vdc}$; $T_A = 0^\circ\text{C} \sim 70^\circ\text{C}$)

特 性	記号	Min	Max	単位
入力 "H" 電圧	V_{IH}	2.0	V_{CC}	V
入力 "L" 電圧	V_{IL}	$V_{SS} - 0.3$	0.8	V
入力リーク電流 @5.25 V	I_{in}	—	2.5 20	μA
スリー・ステート (オフ・ステート) 入力電流 @2.4 V/0.4 V	I_{TTL}	—	20	μA
出力 "H" 電圧 ($I_{OH} = -400\ \mu\text{A}$)	V_{OH}	$V_{CC} - 0.75$ 2.4	— —	V
出力 "L" 電圧 ($I_{OL} = 1.6\text{ mA}$) ($I_{OL} = 3.2\text{ mA}$) ($I_{OL} = 35.0\text{ mA}$) ($I_{OL} = 5.3\text{ mA}$)	V_{OL}	— — — —	0.5 0.5 0.5 0.5	V
消費電力 (クロック周波数=8 MHz)	P_D	—	1.5	W
入力容量 ($V_{in} = 0\text{ V}$, $T_A = 25^\circ\text{C}$; 周波数=1 MHz)	C_{in}	—	10.0	pF

* 470 Ω のプルアップ抵抗を外付

4・2 AC電気的特性 ($V_{CC}=5.0Vdc \pm 5\%$, $V_{SS}=0Vdc$; $T_A=0^\circ C \sim 70^\circ C$)

番号	特 性	記 号	4 MHz, MC68000L4		6 MHz, MC68000L6		8 MHz, MC68000L8		10 MHz, MC68000L10		12.5 MHz, MC68000L12		単位
			Min	Max	Min	Max	Min	Max	Min	Max	Min	Max	
1	Clock Period	t _{CC}	250	500	167	500	125	500	100	500	80	250	ns
2	Clock Width Low	t _{CL}	115	250	75	250	55	250	45	250	35	125	ns
3	Clock Width High	t _{CH}	115	250	75	250	55	250	45	250	35	125	ns
4	Clock Fall Time	t _{CF}	—	10	—	10	—	10	—	10	—	5	ns
5	Clock Rise Time	t _{CR}	—	10	—	10	—	10	—	10	—	5	ns
6	Clock Low to Address	t _{CLAV}	—	90	—	80	—	70	—	55	—	55	ns
6A	Clock High to FC Valid	t _{CHFCV}	—	90	—	80	—	70	—	60	—	55	ns
7	Clock to Address Data High Impedance (Maximum)	t _{CHAZA}	—	120	—	100	—	80	—	70	—	60	ns
8	Clock High to Address/FC Invalid (Minimum)	t _{CHAZA}	0	—	0	—	0	—	0	—	0	—	ns
9	Clock High to \overline{AS} , \overline{DS} Low (Maximum)	t _{CHSLA}	—	80	—	70	—	60	—	55	—	55	ns
10	Clock High to \overline{AS} , \overline{DS} Low (Minimum)	t _{CHSLA}	0	—	0	—	0	—	0	—	0	—	ns
11	Address to \overline{AS} , \overline{DS} (Read)/Low/ \overline{AS} Write	t _{AVSL}	55	—	35	—	30	—	20	—	0	—	ns
11A	FC Valid to \overline{AS} , \overline{DS} (Read)/Low/ \overline{AS} Write	t _{FCVSL}	80	—	70	—	60	—	50	—	40	—	ns
12	Clock Low to \overline{AS} , \overline{DS} High	t _{CLSH}	—	90	—	80	—	70	—	55	—	50	ns
13	\overline{AS} , \overline{DS} High to Address/FC Invalid	t _{SHAZ}	60	—	40	—	30	—	20	—	10	—	ns
14	\overline{AS} , \overline{DS} Width Low(Read)/ \overline{AS} Write	t _{SL}	535	—	337	—	240	—	195	—	160	—	ns
14A	\overline{DS} Width Low (Write)	t _{SL}	285	—	170	—	115	—	95	—	80	—	ns
15	\overline{AS} , \overline{DS} Width High	t _{SH}	285	—	180	—	150	—	105	—	65	—	ns
16	Clock High to \overline{AS} , \overline{DS} High Impedance	t _{CHSZ}	—	120	—	100	—	80	—	70	—	60	ns
17	\overline{AS} , \overline{DS} High to $\overline{R/\overline{W}}$ High	t _{SHRH}	60	—	50	—	40	—	20	—	10	—	ns
18	Clock High to $\overline{R/\overline{W}}$ High (Maximum)	t _{CHRH}	—	90	—	80	—	70	—	60	—	60	ns
19	Clock High to $\overline{R/\overline{W}}$ High (Minimum)	t _{CHRH}	0	—	0	—	0	—	0	—	0	—	ns
20	Clock High to $\overline{R/\overline{W}}$ Low	t _{CHRL}	—	90	—	80	—	70	—	60	—	60	ns
21	Address Valid to $\overline{R/\overline{W}}$ Low	t _{AVRL}	45	—	25	—	20	—	0	—	0	—	ns

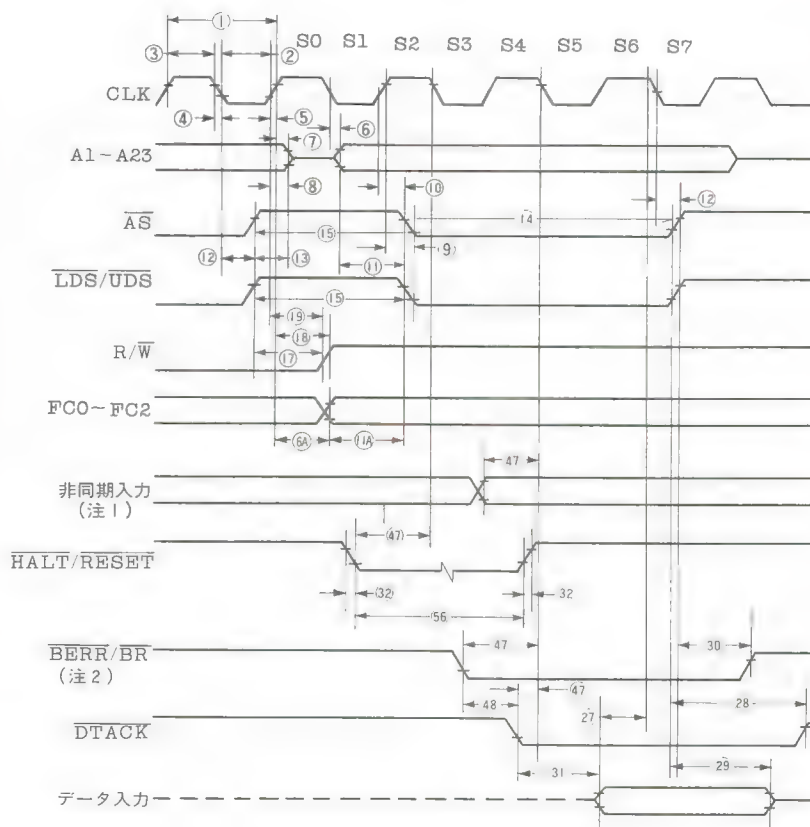
21A ¹	FC Valid to R/W Low	PCVBL	80	—	70	—	60	—	50	—	30	—	ns
22 ²	R/W Low to \overline{DS} Low (Write)	trLSL	200	—	140	—	80	—	50	—	30	—	ns
23	Clock Low to Data Out Valid	tCLOD	—	90	—	80	—	70	—	55	—	55	ns
25 ³	\overline{DS} High to Data Out Invalid	tSDO	60	—	40	—	30	—	20	—	15	—	ns
26 ⁴	Data Out Valid to \overline{DS} Low (Write)	tOOSL	55	—	35	—	30	—	20	—	15	—	ns
27 ⁵	Data In to Clock Low (Setup Time)	tDCL	30	—	25	—	15	—	15	—	15	—	ns
28 ⁶	AS, \overline{DS} High to \overline{DTACK} High	tSDAH	0	240	0	160	0	120	0	90	0	70	ns
29	\overline{DS} High to Data Invalid (Hold Time)	tSDDI	0	—	0	—	0	—	0	—	0	—	ns
30	AS, \overline{DS} High to \overline{BERR} High	tSDHH	0	—	0	—	0	—	0	—	0	—	ns
31 ⁷	\overline{DTACK} Low to Data In (Setup Time)	tDALD	—	180	—	120	—	90	—	65	—	50	ns
32	HALT and \overline{RESET} Input Transition Time	tTRIP ⁸	0	200	0	200	0	200	0	200	0	200	ns
33	Clock High to \overline{BG} Low	tCHGL	—	90	—	80	—	70	—	60	—	50	ns
34	Clock High to \overline{BG} High	tCHGH	—	90	—	80	—	70	—	60	—	50	ns
35	\overline{BR} Low to \overline{BG} Low	tBRLGL	1.5	3.0	1.5	3.0	1.5	3.0	1.5	3.0	1.5	3.0	Clk Per.
36	\overline{BR} High to \overline{BG} High	tBRHGH	1.5	3.0	1.5	3.0	1.5	3.0	1.5	3.0	1.5	3.0	Clk Per.
37	\overline{BTACK} Low to \overline{BG} High	tBALGH	1.5	3.0	1.5	3.0	1.5	3.0	1.5	3.0	1.5	3.0	Clk Per.
38	\overline{BG} Low to Bus High Impedance(With AS High)	tGLZ	—	120	—	100	—	80	—	70	—	60	ns
39	\overline{BG} Width High	tGH	1.5	—	1.5	—	1.5	—	1.5	—	1.5	—	Clk Per.
46	\overline{BGACK} Width	tBGL	1.5	—	1.5	—	1.5	—	1.5	—	1.5	—	Clk Per.
47 ⁹	Asynchronous Input Setup Time	tASH	30	—	25	—	20	—	20	—	20	—	ns
48	\overline{BERR} Low to \overline{DTACK} Low (Note 3)	tBELDAL	50	—	50	—	50	—	50	—	50	—	ns
53	Data Hold from Clock High	tCHDO	0	—	0	—	0	—	0	—	0	—	ns
55	R/W to Data Bus Impedance Change	tRDO	55	—	35	—	30	—	20	—	10	—	ns
56	Halt/ \overline{RESET} Pulse Width (Note 4)	tURPW	10	—	10	—	10	—	10	—	10	—	—

注：

- 50 pF 以下の負荷の場合は、表の値から 5 ns 差し引いた値となる。
- 実際の値はクロック周期により異なる。
- 47 が \overline{DTACK} , \overline{BERR} と共に満足される。48 は 0 ns となる。
- V_{CC} が供給されて 100 ns 後。
- 44A は T6F, BF4, R9M のマスキング・セットでは表の値より 1 クロック周期少ない。
- 非同期セットアップ時間 (47) 要求が満たされると \overline{DTACK} が “L” からデータのセットアップ時間 (43) 要求は無視できる。そのときデータはデータ入力から クロック “L” のセットアップ時間 (47) が満たされていなければならない。

4・3 リード・サイクルのタイミング

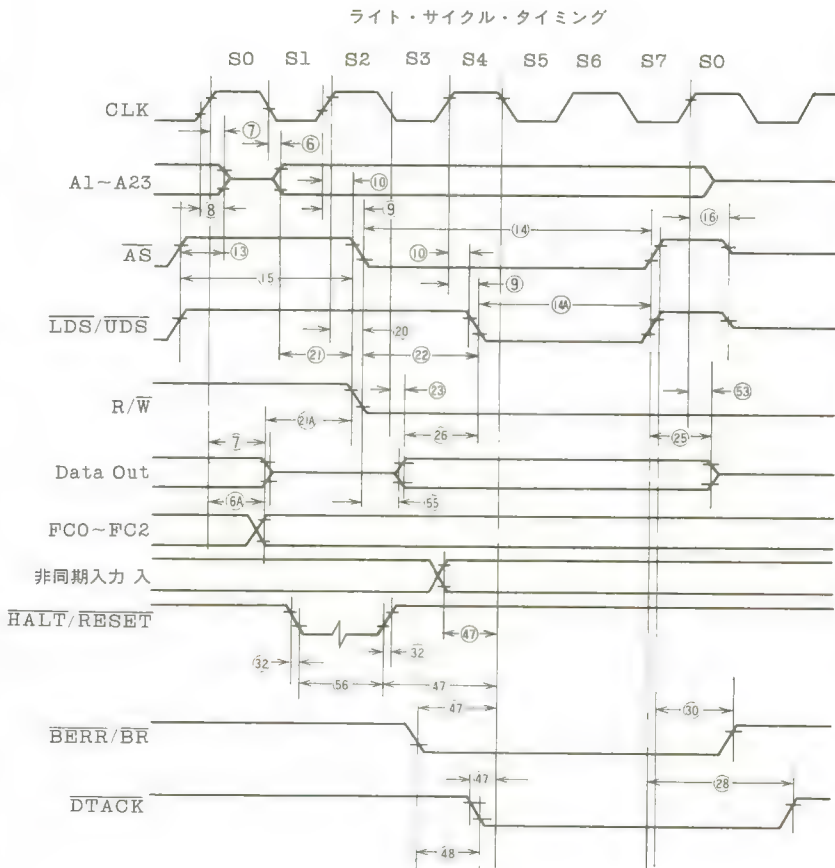
リード・サイクル・タイミング



注：

1. 非同期入力 \overline{BGACK} , $\overline{IPL0} \sim \overline{IPL2}$, \overline{VPA} は次のクロックの立ち下がりで検出されます。そのためにはセット・アップ時間が必要である。
2. \overline{BR} はこのバス・サイクルの終りで認識される必要があるときのみ、このタイミングでアサートされる必要がある。
3. 特記なき場合、すべての入力、出力波形は“H”電圧=2.0V, “L”電圧=0.8Vを測定点として規定される。

4.4 ライト・サイクルのタイミング



注：特記なき場合、すべての入力、出力波形は“H”電圧=2.0V，“L”電圧=0.8Vを測定点として規定される。

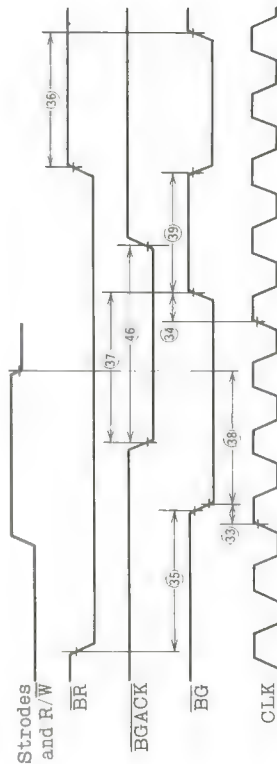
4.5 バス・アービトレーション

AC電気的特性——バス・アービトレーション ($V_{CC}=5.0V_{dc} \pm 5\%$, $V_{SS}=0V_{dc}$, $T_A=0^\circ C \sim 70^\circ C$, 下図参照)

番号	特 性	記 号	4 MHz		6 MHz		8 MHz		10 MHz		12.5 MHz		単 位
			MC 68000L4	Min	Max	MC 68000L6	Min	Max	MC 68000L8	Min	Max	MC 68000L12	
33	Clock High to \overline{BG} Low	t_{CLHL}	—	90	—	80	—	70	—	60	—	50	ns
34	Clock High to \overline{BG} High	t_{CLHH}	—	90	—	80	—	70	—	60	—	50	ns
35	\overline{BR} Low to \overline{BG} Low	t_{BRGL}	1.5	3.5	1.5	3.5	1.5	3.5	1.5	3.5	1.5	3.0	Clk. Per.
36	\overline{BR} High to \overline{BG} High	t_{BRHH}	1.5	3.0	1.5	3.0	1.5	3.0	1.5	3.0	1.5	3.0	Clk. Per.
37	\overline{BGACK} Low to \overline{BG} High	t_{BGLH}	1.5	3.0	1.5	3.0	1.5	3.0	1.5	3.0	1.5	3.0	Clk. Per.
38	\overline{BG} Low to Bus High Impedance (With \overline{AS} High)	t_{GLZ}	—	120	—	100	—	80	—	70	—	60	ns
39	\overline{BG} Width High	t_{GH}	1.5	—	1.5	—	1.5	—	1.5	—	1.5	—	Clk. Per.
46	\overline{BGACK} Width	t_{BGL}	1.5	—	1.5	—	1.5	—	1.5	—	1.5	—	Clk. Per.

——AC電気的波形——バス・アービトレーション——

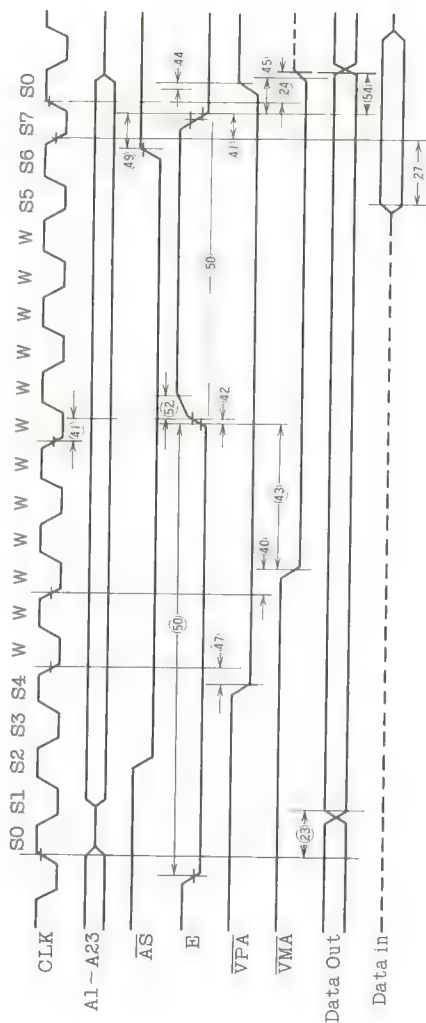
この波形はタイミング規定のエッジからエッジの測定用のものであり入力信号の機能を説明するものではない。他の機能説明とそれに関するデバイス動作の図も参照されたい。



注:

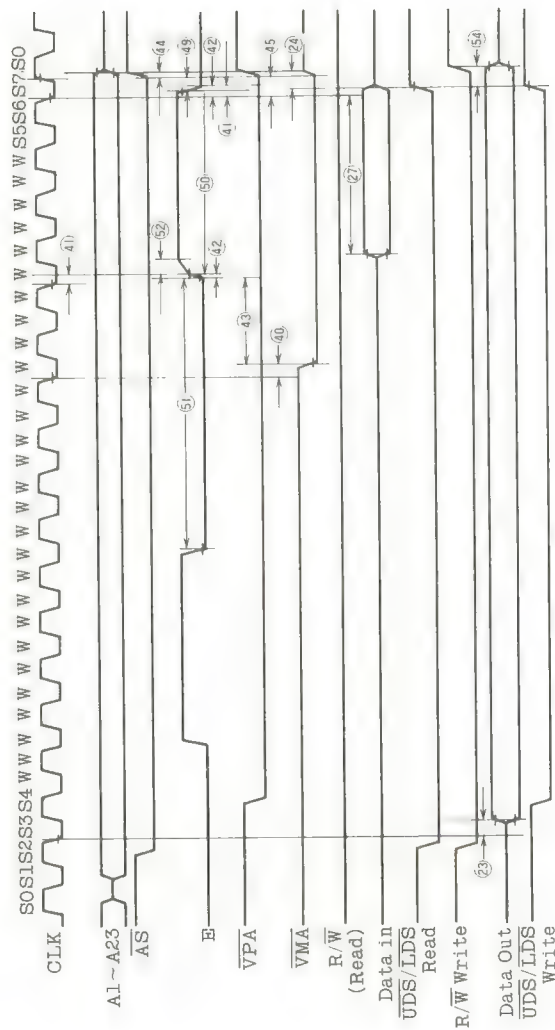
1. 非同期入力 \overline{BER} , \overline{BGACK} , \overline{BR} , \overline{DTACK} , $\overline{IPL0} \sim \overline{IPL2}$, \overline{VPA} は次のクロックの立ち下がりがて検出される。そのためには、セットアップ時間が必要である。
2. すべての入力、出力の波形は“H”電圧=2.0V, “L”電圧=0.8Vを測定点として規定される。

M6800タイミング・ベスト・ケース



注：この図はM6800の最良のタイミングを示し、Eの立ち下がり後3つ目のシステム・クロック・サイクルの前にVPAが下がっている。

M6800 タイミング・ケース



参 考 文 献

(モトローラ社提供)

- 1) MC 68000 データブック
- 2) MC 68000 マイクロプロセッサ・ユーザーズ・マニュアル
- 3) M 68000 リアルタイム・マルチタスキング・ソフトウェア・ユーザーズ・ガイド
- 4) M 68000 EXORmacs 開発システム操作マニュアル
- 5) スピード・マスタ 68 K 取扱い説明書
- 6) M 68000 ユーザ・システム・エミュレータ・ユーザーズ・ガイド
- 7) M 68000 常駐ストラクチャード・アセンブラ・リファレンス・マニュアル
- 8) VERSAdos I/O およびファイル管理サービス・マニュアル
- 9) M 68000/IPC コマンドチャンネル・ソフトウェア・インターフェース・リファレンス・マニュアル
- 10) M 68000 リンケージ・エディタ・ユーザーズ・マニュアル
- 11) マルチ・チャンネル・コミュニケーション・モジュール・ユーザーズ・ガイド
- 12) M 68 KBSA リアルタイム・バス・アナライザー・ユーザーズ・ガイド

索引

[ア 行]

後入れ先出し	28
アセンブラ	12
アドレス・エア	88
アドレス空間	148
アドレス・ストローブ	36
アドレス・バス	34
アドレス・レジスタ	18
アドレス・レジスタ間接	58
アドレス・レジスタ直接	56
アドレッシング・モード	54
アブソリュート・ショート	64
アブソリュート・モード	64
アブソリュート・ロング	64
アレイ・チェーン・モード	155
アンリンク命令	129
I/O 機能	12
違法命令	94
イミディエート・モード	68
インタフェース	159
インデックス付きモード	66
インプライド・モード	70
MMU	149

MPU コントロール・シグナル	44
MPU ステータス・シグナル	48
SR	20
エクサマクス	6, 168
エクセプション・プロセッシング	84
オフセット付きアドレス・レジスタ間接	61, 62
オフセット付きモード	66

[カ 行]

開発支援装置	6
外部起因	74
仮想マシン	166, 212
逆アセンブラ	12
キュー	30
キューの構成	30
クイック・イミディエート	68
クロック入力	48
ゲート・ポインタ	30
コンティニュー・モード	154
最小システム	136

[サ 行]

CRT エディタ	188
システム・コントロール.....	44
システム・スタック.....	28
システム制御命令	132
システム・タスク	200
システムの基本構成	137
システム・パッケージ	185
実効アドレス.....	54
シフト命令	115
上位/下位データ・ストローブ	36
上位システム構成	139
シングル・ステップ・モード.....	47
シンボリック・デバッカ	191

スタックの構成.....	28
スタック・ポインタ.....	18
ステータス・レジスタ.....	20
ストラクチャード・マクロ・アセンブラ.....	188

整数算術演算命令	107
絶対モード.....	64
セマフォ	201

[タ 行]

中位システム構成	138
DIP	2
DMA コントローラ	152
データ移動命令	102
データ構成.....	26
データ転送.....	50
データ転送アクノリッジ.....	36

データ・バス.....	34
データ・レジスタ.....	18
データ・レジスタ直接.....	56
デバッグ.....	12
同期式バス・コントロール.....	51
特権違反.....	94
特権状態.....	92
トラップ.....	90
トレース.....	94

[ナ 行]

内部起因.....	74
二重マップ方式	194

[ハ 行]

PASCAL コンパイラ	190
バーサ・バス	140
バス・アービトレーション.....	41
バス・アービトレーション・コントロール.....	36
バス・エラー.....	86
バス・エラー例外シーケンス.....	86
バス・グラント	36, 41
バス・コントロール・シグナル.....	36
バス・サイクルのリラン.....	86
バス・シグナル.....	34
バス・マスタの応答.....	41
バス・リクエスト.....	36
バス要求.....	41
バーチャル機構	146
バリッド周辺アドレス.....	46
バリッド・メモリ・アドレス.....	46

PC	20
BCD 演算命令	119
ビット操作命令	121
非同期バス・コントロール	36, 50
ピン構成	16

ファームウェア	12
ファンクション・モード	48
フォーマット	192
プット・ポインタ	30
プリデクリメント	30, 61
プログラム・カウンタ	20
プログラム・カウンタ相対モード	66
プログラム制御命令	124
プロセッサのステータス	48

ベクタ番号のセネレーション	78
ページング・セグメンテーション	24

FORTRAN コンパイラ	190
ポストインクリメント	30, 60

[マ 行]

命令形式	100
命令タイプ	100
命令フォーマット	100
メモリ管理ユニット	148
メモリの構成	24
メモリ・マップ	25
モード・フィールド	54
モニタ	12

モノボード・マイクロコンピュータ	144
------------------------	-----

[ヤ 行]

ユーザ・スタック	28
ユーザ・タスク	200

[ラ 行]

ライト・サイクル	50
リセット	90
リード・サイクル	50
リード/ライト	36
リンク・アレイ・チェーン・モード	155
リンク命令	129
リンケージ・エディタ	190

例外処理	74
例外処理シーケンス	84
レジスタ間接モード	58
レジスタ直接モード	56
レジスタの構成	18
レジスタ・フィールド	54

68 系コントロール	44
68000 の基本システム	137
ローティット命令	115
論理操作命令	113

[ワ 行]

ワードの構成	24
割込みコントロール	44

著者略歴

昭和42年 法政大学工学部電気工学科卒
現在 C & IT (株)代表取締役社長
著書 「パソコン入門心得帖」(共著;
オーム社)

図解16ビットマイクロコンピュータ MC 68000 の 使 い 方

© 小島 進 1983

昭和58年10月1日 第1版第1刷発行
平成4年5月20日 第1版第7刷発行

OHM・OHM・OHM・OHM
著者承認
検印省略
OHM・OHM・OHM・OHM

著者	こ じ ま 小 島 進
発行者	株式会社 オーム社 代表者 佐藤 政次
発行所	株式会社 オーム社 郵便番号 101 東京都千代田区神田錦町3-1 振替 東京6-20018 電話 03(3233)0641(代表)

Printed in Japan

印刷 中央印刷 製本 関川製本所
落丁・乱丁本はお取替えいたします

ISBN 4 - 274 - 07162 - 6



スーパービギナーシリーズ

■ B5 変形判

■ 2色刷



一太郎 Ver.4 入門

㈱コンピュータバンク 監修
高橋美穂 著 256 頁

MS-DOS 入門

茅野昌明 著 250 頁

松 Ver.5 入門

小林宏之 著 290 頁

1-2-3 Notebook 入門

吉木一彦 著 254 頁

桐 Ver.3 入門

梓 史朗 著 260 頁

MS-Works 入門

竹内 瞳 著 240 頁

■ ■ ■ ■ これならわかる ■ ■ ■ ■

パーソナルブック

だれでも使える Windows 3.0

島谷明男 著 B5判 274 頁

C 言語によるプログラミング — 基礎 編 —

内田智史 編著 B5判 358 頁

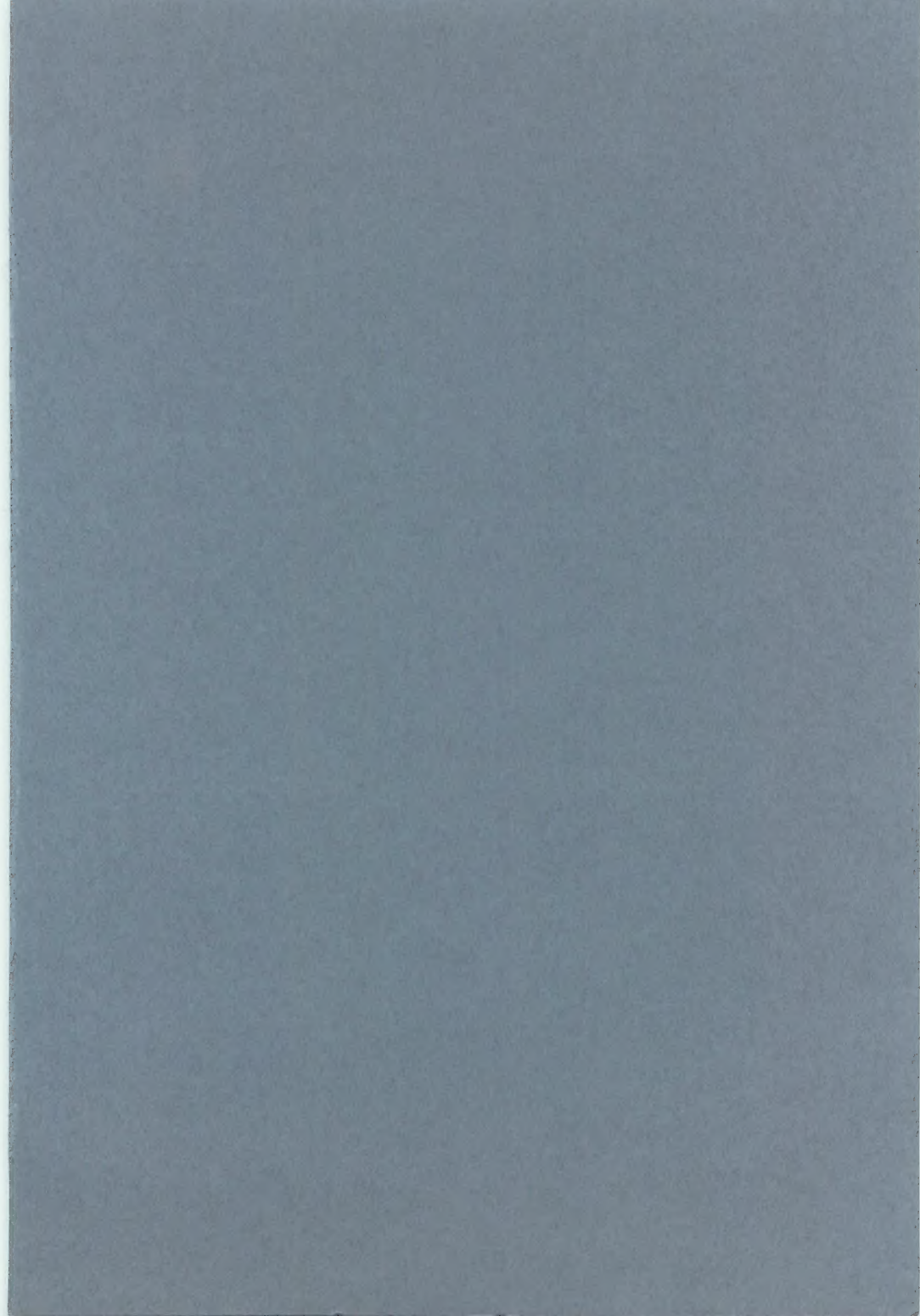
Sun ユーザのための やさしい UNIX のはじめかた

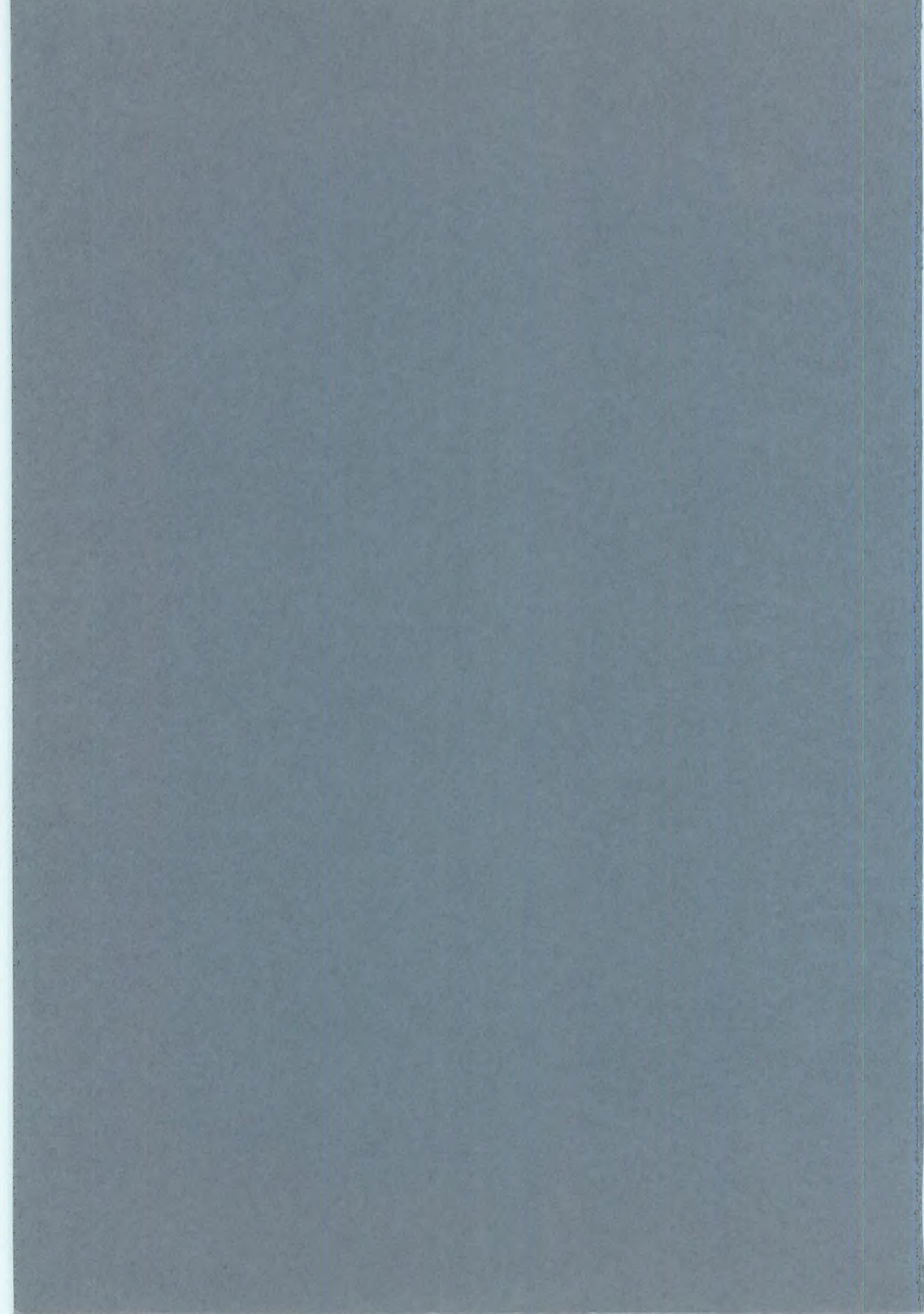
三上市蔵 編 A5判 292 頁

C 言語によるプログラミング — 応用 編 —

内田智史 編著 B5判 350 頁

Ohmsha
Ohmsha





図解コンピュータシリーズ

江村親朗 監修

コンピュータシステム入門

(改訂2版) 江村親朗・野津 昭 共著 (A5・p.268)

ハードウェア・システム入門

江村親朗 著 (A5・p.216)

FORTRAN 入門

番老沢成享 著 (A5・p.230)

COBOL 入門

番老沢成享 著 (A5・p.350)

PL/I 入門

光吉民恵 著 (A5・p.244)

PASCAL プログラミング入門

三沢常男・市川隆男 共著 (A5・p.264)

BASIC プログラミング入門

平山静夫 著 (A5・p.290)

アセンブラプログラミング入門

瀬下孝之・前田忠彦 共著 (B5・p.380)

APL 入門

金子修弘 著 (A5・p.236)

日本語 APL 入門

平尾隆行 著 (A5・p.236)

簡易照会言語入門

—関係データベースシステム—

平尾隆行 著 (A5・p.228)

ストラクチャード・プログラミング入門

(改訂2版) 國友義久 著 (A5・p. 258)

オペレーティング・システム入門

江村親朗 著 (A5・p.176)

ファイル編成入門

山谷正己 著 (A5・p.184)

データベース入門

(改訂2版) 穂積良介 著 (A5・p.228)

仮想記憶システム入門

山谷正己 著 (A5・p.140)

エキスパートシステム入門

平尾隆行 著 (A5・p.176)

情報通信システム入門

八島朝一 著 (A5・p.244)

データ通信システム入門

(改訂2版) 保坂岩男 原著・石坂充弘 著 (A5・p.270)

EDP システム設計入門

南條 義 著 (A5・p.204)

オフィスコンピュータ入門

魚田勝臣・富沢研三 共著 (A5・p.202)

オフィスオートメーション入門

(改訂2版) 中村 茂 著 (A5・p.238)

RPG プログラミング技法

野口正雄 著 (B5・p.232)

プログラム流れ図の作成技法

江村親朗・野津 昭 共著 (B5・p.358)

対話式計算システムの活用技法

平尾隆行 著 (B5・p.262)

プログラム開発管理

(改訂2版) 國友義久 著 (B5・p.218)

仮想計算機システム

山谷正己 著 (B5・p.304)

多重仮想記憶オペレーティングシステム

鎌田 肇 著 (B5・p.228)

コンピュータとアプリケーション

寺沢康夫 著 (B5・p.236)

データベース/データ通信プログラミング

平尾隆行 著 (B5・p.220)

ISBN4-274-07162-6 C3055 P3000E 定価 3000 円 (本体 2913 円)